



Universidade do Estado do Rio de Janeiro
Centro de Tecnologia e Ciências
Instituto de Matemática e Estatística


Bruno Porto Masquio

Emparelhamentos Desconexos

Rio de Janeiro
2019

Bruno Porto Masquio

Emparelhamentos Desconexos



Dissertação apresentada como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Ciências Computacionais, da Universidade do Estado do Rio de Janeiro.

Orientador: Prof. Dr. Paulo Eustáquio Duarte Pinto

Orientador: Prof. Dr. Jayme Luiz Szwarcfiter

Rio de Janeiro

2019

Bruno Porto Masquio

Emparelhamentos Desconexos

Dissertação apresentada como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Ciências Computacionais, da Universidade do Estado do Rio de Janeiro.

Aprovada em 13 de Março de 2019
Banca Examinadora:

Prof. Dr. Paulo Eustáquio Duarte Pinto (Orientador)
Instituto de Matemática e Estatística - UERJ

Prof. Dr. Jayme Luiz Szwarcfiter (Orientador)
Instituto de Matemática e Estatística - UERJ

Prof. Dr. Fabiano de Souza Oliveira
Instituto de Matemática e Estatística - UERJ

Prof. Dr. Diana Sasaki de Souza Pereira
Instituto de Matemática e Estatística - UERJ

Prof. Dr. Uéverton dos Santos Souza
Universidade Federal Fluminense - UFF

Rio de Janeiro
2019

RESUMO

MASQUIO, Bruno Porto. *Emparelhamentos Desconexos*. 2019. Dissertação (Mestrado em Ciências Computacionais) – Instituto de Matemática e Estatística, Universidade do Estado do Rio de Janeiro, Rio de Janeiro.

A determinação de emparelhamentos máximos em grafos corresponde a problemas há muito tempo estudados e que trazem grandes contribuições à Teoria de Grafos, tanto do ponto de vista teórico quanto prático. Como tradição, há numerosos estudos na área destinados a emparelhamentos irrestritos e sem pesos. Mais recentemente, foram propostos os emparelhamentos restritos a subgrafos, que consideram propriedades dos subgrafos induzidos pelos vértices do emparelhamento. Neste trabalho, abordamos uma dessas novas propostas, que são os emparelhamentos desconexos, os quais procuram estudar emparelhamentos máximos tais que o subgrafo induzido pelos vértices do emparelhamento seja desconexo. Conseguimos resolver o problema para classes simples de grafos tais como caminhos, ciclos, completos. Além disso, também obtivemos resultados para grafos split, árvores, grafos bloco e grafos cordais. Para essas três últimas classes, apresentamos uma base teórica que inclui teoremas de caracterizações, assim como três novos algoritmos. O trabalho também inclui um quarto algoritmo que determina um emparelhamento máximo para grafos bloco em tempo linear.

Palavras-chave: Algoritmos. Árvores. Emparelhamento. Emparelhamento Desconexo. Grafo Bloco. Grafo Split. Grafo Cordal. Separadores minimais.

ABSTRACT

MASQUIO, Bruno Porto. *Disconnected Matchings*. 2019. Dissertação (Mestrado em Ciências Computacionais) – Instituto de Matemática e Estatística, Universidade do Estado do Rio de Janeiro, Rio de Janeiro.

Graph matchings are problems that have been studied for a long time and which bring great contributions to Graph Theory from both the theoretical and practical points of view. As a tradition, there are numerous studies in this field for unrestricted and unweighted matchings. More recently, subgraph-restricted matchings have been proposed, which consider properties from the subgraph induced by the matching vertices. In this paper, we approach one of these new proposals, which are disconnected matchings, which seeks to study matchings and its cardinalities, such that the subgraph induced by matching vertices is disconnected. We were able to solve the problem for simple classes of graphs like paths, cycles and complete. In addition, we obtained results for split graphs, trees, block graphs and chordal graphs. For these last three classes, we present a theoretical basis that includes theorems of characterizations, as well as three new algorithms. This paper also includes a fourth algorithm, which finds a maximum matching for block graphs in linear time.

Keywords: Algorithms. Block Graphs. Disconnected Matching. Matching. Split Graphs. Chordal Graphs. Trees. Minimal separators.

LISTA DE ILUSTRAÇÕES

Figura 1 – Representação geométrica de um grafo	13
Figura 2 – O grafo K_5	15
Figura 3 – A estrela S_5 e a estrela dupla $S_{4,3}$	16
Figura 4 – Um grafo split	17
Figura 5 – Uma árvore e uma floresta	18
Figura 6 – Um grafo com um único separador minimal e dois separadores de vértices minimais	19
Figura 7 – Um grafo bloco	21
Figura 8 – Um grafo cordal e um grafo não cordal	22
Figura 9 – Um grafo cordal	24
Figura 10 – Uma clique tree correspondente ao grafo cordal da Figura 9	25
Figura 11 – Um digrafo	26
Figura 12 – Matrizes de adjacências	27
Figura 13 – Matriz de incidências	28
Figura 14 – Três emparelhamentos em grafos	32
Figura 15 – Caminhos aumentante(a) e alternante(b)	33
Figura 16 – Um grafo de exemplo	34
Figura 17 – Um emparelhamento desconexo, induzido, unicamente restrito e acíclico no grafo da Figura 16	34
Figura 18 – Um emparelhamento acíclico, conexo, unicamente restrito e livre de iso- lamentos no grafo da Figura 16	35
Figura 19 – Um emparelhamento desconexo no grafo da Figura 16	36
Figura 20 – O caminho de comprimento 6	39
Figura 21 – O caminho de comprimento 5	39
Figura 22 – Os ciclos C_6 (a) e C_7 (b)	40
Figura 23 – Um exemplo de árvore T	45
Figura 24 – Grafos induzidos por emparelhamentos desconexos máximos em T	46
Figura 25 – Um grafo bloco e sua árvore auxiliar	49
Figura 26 – A árvore auxiliar $T(B)_{v_{13}}^{v_{12}}$ e o subgrafo induzido $B_{v_{13}}^{v_{12}}$ do grafo bloco da Figura 25	51
Figura 27 – Grafo induzido por um emparelhamento desconexo máximo em B	56
Figura 28 – Um grafo cordal e uma clique tree correspondente	66
Figura 29 – Dois grafos induzidos por vértices de emparelhamentos desconexos má- ximos do grafo da Figura 28a	68

LISTA DE TABELAS

Tabela 1 – Tabela com o conteúdo das listas após a execução de $BUSCA1(v_1)$. . .	45
Tabela 2 – Tabela com o conteúdo das listas após a execução de $BUSCA2(v_1)$. . .	46
Tabela 3 – Tabela com o conteúdo das listas após a execução de $BUSCA1(v_{12})$. . .	55
Tabela 4 – Tabela com o conteúdo das listas após a execução de $BUSCA2(v_{12})$. . .	56
Tabela 5 – Tabela com o conteúdo de emparelhamentos novos encontrados após a execução da busca	60
Tabela 6 – Tabela com o conteúdo das listas após a execução de $BUSCA1(a, a)$. . .	66
Tabela 7 – Tabela com o conteúdo das listas após a execução de $BUSCA1(a, a)$. . .	67
Tabela 8 – Tabela com o conteúdo das listas após a execução de $BUSCA2(a, a)$. . .	67
Tabela 9 – Tabela com o conteúdo das listas após a execução de $BUSCA2(a, a)$. . .	67

LISTA DE ALGORITMOS

Algoritmo 1 – Busca em profundidade (recursivo)	30
Algoritmo 2 – Busca em largura	30
Algoritmo 3 – Cardinalidade do Emparelhamento Desconexo máximo em uma árvore	43
Algoritmo 4 – Cardinalidade do Emparelhamento Desconexo máximo em um grafo bloco	54
Algoritmo 5 – Emparelhamento máximo em um grafo bloco	59
Algoritmo 6 – Emparelhamento desconexo máximo em um grafo cordal	64

SUMÁRIO

	INTRODUÇÃO	9
1	CONCEITOS TEÓRICOS	11
1.1	Complexidade Computacional	11
1.1.1	Notação O-grande	11
1.1.2	Notação Theta	12
1.1.3	Problema de Decisão	12
1.1.4	Classes de Problemas	12
1.2	Teoria de grafos	13
1.3	Árvores	17
1.4	Conectividade	19
1.5	Grafos Bloco	21
1.6	Grafos Cordais	21
1.7	Grafos Direcionados	26
1.8	Representação de Grafos	27
1.9	Buscas em grafos	29
1.9.1	Busca em profundidade	29
1.9.2	Busca em largura	30
2	EMPARELHAMENTOS	31
2.1	Emparelhamentos restritos a subgrafos	33
3	EMPARELHAMENTOS DESCONEXOS	37
3.1	Emparelhamento desconexo em caminhos	38
3.2	Emparelhamento desconexo em ciclos	39
3.3	Emparelhamento desconexo em grafos split	40
3.4	Emparelhamento desconexo em árvores	40
3.5	Emparelhamento desconexo em grafos bloco	47
3.5.1	Emparelhamento máximo em grafos blocos	58
3.6	Emparelhamento desconexo em grafos cordais	61
	CONCLUSÕES E TRABALHOS FUTUROS	71
	REFERÊNCIAS	74

INTRODUÇÃO

Um emparelhamento em grafos é um conjunto de arestas não adjacentes duas a duas. Os problemas de emparelhamentos máximos em grafos são amplamente estudados há décadas pela comunidade científica e já possuem diversos resultados e aplicações importantes.

Esses estudos abordam o problema de diversas formas, como, por exemplo, para grafos ponderados ou não, para classes de grafos específicas e para emparelhamentos restritos a subgrafos. Essa última abordagem, recentemente proposta, procura resolver o problema tal que certas propriedades de subgrafos induzidos pelos vértices de um emparelhamento sejam válidas.

Algumas das propriedades que esses subgrafos podem ter incluem que os mesmos sejam, por exemplo, 1-regular, acíclico, conexo ou desconexo. O problema do emparelhamento envolvendo essa última propriedade, ou seja, de que o grafo induzido pelos vértices do emparelhamento seja desconexo, é o foco deste trabalho.

A determinação de emparelhamentos desconexos pode ajudar resolver problemas que envolvem segurança e sigilo de informações. Por exemplo, suponha uma empresa de produção de software deseja formar equipes de desenvolvedores os quais utilizam a prática de programação em par, originada do desenvolvimento ágil. Além disso, é desejado que as equipes sejam isoladas de forma que ninguém de equipes distintas tenha tido contato anterior. A partir da relação de todas as pessoas que já tiveram contato, podemos formar um grafo e, assim, determinar emparelhamentos desconexos para resolver o problema. Nesse caso, as equipes seriam representadas por componentes conexas do subgrafo e os pares de programadores, por arestas do emparelhamento.

Neste trabalho, foram realizados estudos teóricos, incluindo teoremas e caracterizações do emparelhamento desconexo para algumas classes importantes de grafos. Alguns desses resultados poderiam ser aplicados a grafos gerais, mas ainda sem garantia de obtenção de algoritmos polinomiais. Também apresentamos, acompanhados de suas provas de corretude e de complexidade de tempo, quatro novos algoritmos. Três deles determinam emparelhamentos desconexos máximos em árvores, em grafos bloco e em grafos cordais. O quarto algoritmo obtém emparelhamentos máximos em grafos bloco, com complexidade de tempo linear.

Nos algoritmos para emparelhamentos desconexos máximos mencionados, foi desenvolvida uma técnica usada de forma semelhante para as três classes. Essa técnica faz uso de uma representação do grafo em forma de uma árvore especial, cuja estrutura possa identificar os separadores minimais. Ao enraizarmos essas árvores, visitamos os vértices em duas buscas em profundidade, de modo que a primeira processa os vértices das folhas à raiz e a segunda, da raiz às folhas. Para árvores, a representação é o próprio grafo. Para grafos bloco, foi criada uma representação especial e, para grafos cordais, foi utilizada a clique tree, uma representação de há muito estabelecida.

Estrutura

O trabalho é organizado da seguinte forma:

- Capítulo 1: descreve a fundamentação teórica da dissertação com conceitos de teoria da computação e teoria dos grafos. Esse capítulo detalha e ilustra estudos de complexidade computacional, classes de problemas, definições e propriedades básicas de grafos, definição e caracterização de algumas classes de grafos, apresentação de alguns problemas clássicos de grafos, caracterizações de árvores, de grafos bloco, de grafos cordais, propriedades e definições de conectividade, definições de digrafos, representações de grafos e suas técnicas computacionais associadas, além da apresentação das teorias e dos algoritmos de duas das principais buscas em grafos.
- Capítulo 2: descreve o problema de emparelhamentos em grafos. Nesse capítulo, são apresentados definições, exemplos e resultados já obtidos na área de emparelhamentos, como teoremas e caracterizações. Além disso, é apresentada a abordagem de emparelhamentos restritos a subgrafos, que categoriza emparelhamentos a partir do subgrafo induzido pelos vértices incidentes do mesmo. As definições desses emparelhamentos são apresentadas e detalhadas. Também são apresentados alguns dos resultados atuais sobre o tema na comunidade científica.
- Capítulo 3: detalha o emparelhamento desconexo e apresenta os resultados obtidos. Esse capítulo apresenta, com detalhes, definições, exemplos e resultados para emparelhamentos desconexos. Primeiramente, são mostrados resultados para grafos simples, como caminhos e ciclos. A seguir, são apresentados os resultados e caracterizações de algumas classes de grafos: grafos split, árvores, grafos bloco e grafos cordais. Para essas últimas três classes, são apresentados teoremas e algoritmos para a resolução do problema. Além disso, é apresentado um algoritmo para emparelhamento máximo em grafos bloco.
- Por fim, apresentamos as conclusões da dissertação em conjunto com as propostas de trabalhos futuros.

1 CONCEITOS TEÓRICOS

Neste capítulo, serão descritos alguns conceitos de Complexidade Computacional e de Teoria dos Grafos, tendo como referência os livros (SZWARCFITER; MARKENZON, 2010; SZWARCFITER, 2018).

1.1 Complexidade Computacional

A teoria da complexidade computacional é o estudo que classifica um problema computacional de acordo com sua dificuldade inerente. A análise da *complexidade de um algoritmo* está ligada diretamente com sua eficiência, de maneira que nos mostre o quanto de um determinado recurso de máquina o algoritmo requer quando executado, em função do tamanho da entrada processada. Geralmente, a complexidade é a medição do tempo de processamento ou consumo de memória. A *complexidade de tempo* se refere à quantidade de instruções primitivas da máquina que um algoritmo requer para sua execução completa. A *complexidade de espaço ou memória* é a análise da quantidade de células de memória que são utilizadas simultaneamente durante a execução do algoritmo.

1.1.1 Notação O-grande

Existem diversas notações e maneiras de classificar a complexidade de um determinado recurso. Uma das classificações fundamentais é a de *pior caso*. O objetivo desta classificação é medir a utilização máxima do recurso em questão. Para medir a complexidade de um algoritmo é necessário associar o parâmetro relativo ao tamanho da entrada, denotado por n , com o recurso computacional da máquina requerido. Este processo é feito através de uma função $f(n)$, porém, na maior parte dos casos, é difícil ou até mesmo impossível obter esta função. Assim, utilizamos a notação $T(f(n))$ para denotar o termo de maior crescimento assintótico da função $f(n)$. Sabemos que $T(f(n))$ se aproxima do valor $f(n)$ com o crescimento da entrada. Desta maneira, temos que:

$$\lim_{n \rightarrow \infty} \frac{f(n)}{T(f(n))} = 1, \text{ onde } T(f(n)) \text{ termo de maior crescimento assintótico da função } f(n).$$

Assim, temos uma família de notações que são amplamente utilizadas para evidenciar a ordem de grandeza deste termo de maior grau da função $f(n)$.

Sejam as funções reais positivas f e g . Dizemos que $f(n) = O(g(n))$ se, para alguma constante positiva C e um valor inicial n_0 , $\forall n \geq n_0$ temos $f(n) \leq Cg(n)$.

Na prática as funções escolhidas para $g(n)$ possuem apenas um termo que representa a ordem da função $f(n)$. Por exemplo, para quaisquer inteiros positivos A , B e C , se $f(n) = An^3 + Bn^2 + Cn$, então $f(n) = O(n^k)$ com $k \geq 3$. Esta notação é amplamente difundida e utilizada na literatura.

Convencionou-se considerar como *tratáveis* os problemas para os quais a complexidade de pior caso pode ser limitada superiormente como um polinômio em n e *intratáveis* em caso contrário.

1.1.2 Notação Theta

Uma das notações utilizadas nesta dissertação para analisar a complexidade de pior caso dos métodos propostos é o *big-O* (O-grande). A outra é a notação Θ . Esta notação é utilizada para classificar a complexidade através de uma função que define seu limite inferior e superior, restringindo a utilização do recurso em questão. Assim, temos a seguinte definição:

Sejam as funções reais positivas f e g . Dizemos que $f(n) = \Theta(g(n))$ se, para duas constantes positivas C_1, C_2 e um valor inicial $n_0, \forall n \geq n_0$, temos $f(n) \geq C_1g(n)$ e $f(n) \leq C_2g(n)$.

Por exemplo, para qualquer inteiro A , se $f(n) = An^4$, então $f(n) = \Theta(n^4)$.

1.1.3 Problema de Decisão

O problema P é um problema de decisão quando o conjunto solução S para qualquer instância é composto somente pelos elementos *Sim* e *Não*. Por exemplo:

- Sejam o número A e um conjunto C de números. Existe algum número $B \in C$ maior que A ?
- Sejam G um mapa de rotas entre cidades, $A \in G$ uma cidade e B um número. É possível visitar todas as cidades de G com uma rota de comprimento menor que B começando pela cidade A ?

1.1.4 Classes de Problemas

Os problemas computacionais são classificados de acordo com sua dificuldade inerente. Portanto, diversas classes são definidas para identificar o quão eficiente um determinado algoritmo pode ser para dado problema de acordo com um recurso computacional. Algumas das classes fundamentais são definidas abaixo:

- Classe P (Polynomial Time): consiste dos problemas de decisão para os quais existe pelo menos um *algoritmo determinístico* com *complexidade polinomial* que o resolve. Ou seja, são os problemas de decisão em que há uma máquina de *Turing* determinística que resolve o problema em um número de passos polinomial em relação ao tamanho da entrada.
- Classe NP (Nondeterministic Polynomial Time): é a classe fundamental que consiste dos problemas de decisão em que a resposta *Sim* pode ser verificada de forma eficiente. A classe pois inclui exatamente os problemas de decisão que admitem algoritmo não determinístico polinomial. Isto é, para qualquer instância que responde *Sim*, a resposta pode ser provada por um certificado que pode ser verificado por uma máquina de *Turing* determinística em um número de passos polinomial em função do seu tamanho.

- Classe *NP-Difícil*: é a classe que consiste dos problemas P tal que todo problema $P' \in NP$ pode ser reduzido em um número de passos polinomial para algum problema pertencente a P . Ou seja, podemos transformar, sempre que existir, uma solução de P' em uma solução de P através de uma máquina de *Turing* determinística com um número de passos polinomial em relação ao seu tamanho. Desta maneira, resolver P' não é mais difícil que resolver P .
- Classe *NP-Completo*: é a classe que consiste dos problemas de decisão que pertencem às classes NP e NP-Difícil.

1.2 Teoria de grafos

Um *grafo* $G(V,E)$ é um conjunto finito não-vazio V e um conjunto E de pares não ordenados de elementos distintos de V . Dizemos que G é *trivial* quando $|V| = 1$. Quando necessário, se utiliza o termo *grafo não direcionado*, para designar um grafo. Os elementos de V são os *vértices* e os de E são as *arestas* de G , respectivamente. Cada aresta $e \in E$ será denotada pelo par de vértices $e = (v,w)$ que a forma. Nesse caso, os vértices v, w são os *extremos* (ou *extremidades*) da aresta e , sendo denominados *adjacentes*. A aresta e é dita *incidente* a ambos v, w . Duas arestas que possuem um extremo comum são chamadas de *adjacentes*. Utilizaremos a notação $n = |V|$ e $m = |E|$.

Um grafo pode ser visualizado através de uma *representação geométrica*, na qual seus vértices correspondem a pontos distintos do plano em posições arbitrárias, enquanto que a cada aresta (v,w) é associada uma linha arbitrária unindo os pontos correspondentes a v, w . Para maior facilidade de exposição, é usual confundir-se um grafo com a sua representação geométrica. Isto é, no decorrer do texto será utilizado o termo *grafo*, significando também a sua representação geométrica.

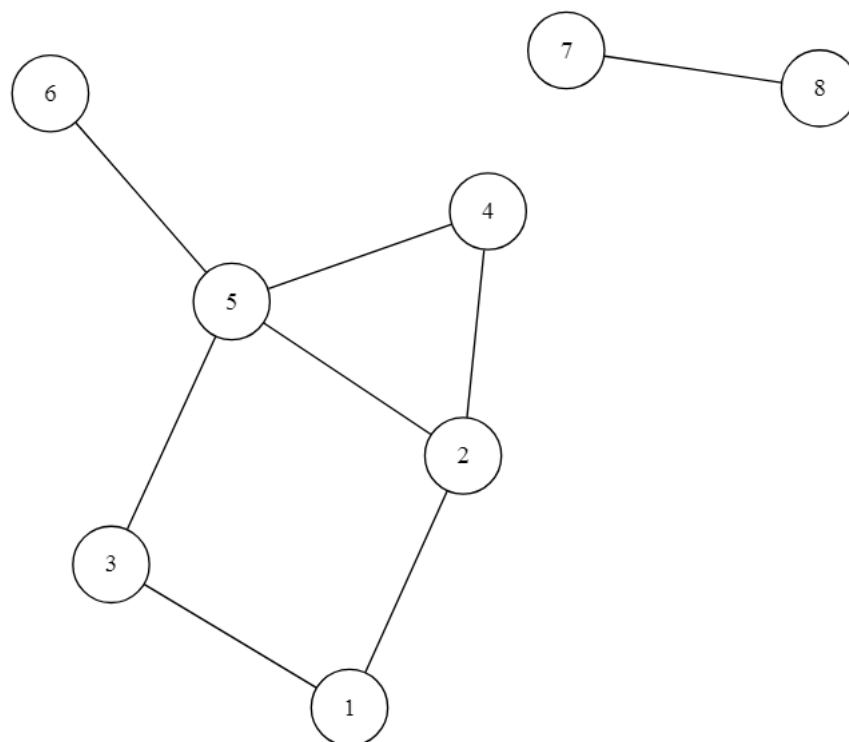


Figura 1 – Representação geométrica de um grafo

A partir desta definição é possível formular o seguinte problema. Dadas duas representações geométricas, correspondem elas a um mesmo grafo? Em outras palavras, é possível fazer coincidir, respectivamente, os pontos de duas representações geométricas, de modo a preservar adjacência (ou seja, de modo a fazer também coincidir as arestas)? Formalmente, o problema pode ser enunciado da seguinte maneira. Dados dois grafos $G_1(V_1, E_1)$, $G_2(V_2, E_2)$, com $|V_1| = |V_2| = n$, existe uma função unívoca $f : V_1 \rightarrow V_2$, tal que $(v, w) \in E_1$ se e somente se $(f(v), f(w)) \in E_2$, para todo $v, w \in V_1$? Em caso positivo, G_1 e G_2 são ditos *isomorfos entre si*. Esse problema, denominado *isomorfismo de grafos*, pode naturalmente ser resolvido por força bruta, examinando-se cada uma das $n!$ permutações de V_1 (ou seja, cada função f possível). Esse algoritmo necessita de pelo menos $\Omega(n!)$ passos, no pior caso. É desconhecido se existe ou não algum algoritmo eficiente para o problema geral de isomorfismo de grafos.

Em um grafo $G(V, E)$, define-se *grau* de um vértice $v \in V$, denotado por $\text{grau}(v)$, como sendo o número de vértices adjacentes a v . Um grafo é *regular* de *grau* r , quando todos os seus vértices possuírem o mesmo grau r . Um vértice que possui grau zero é chamado *isolado*.

Uma sequência de vértices v_1, \dots, v_k tal que $(v_j, v_{j+1}) \in E$, $1 \leq j \leq k-1$, é denominado *caminho* ou *passeio* de v_1, \dots, v_k . Diz-se então que v_1 *alcança* ou *atinge* v_k . Um caminho de k vértices é formado por $k-1$ arestas $(v_1, v_2), (v_2, v_3), \dots, (v_{k-1}, v_k)$. O valor $k-1$ é o *comprimento* do caminho. Se todos os vértices do caminho v_1, \dots, v_k forem distintos, a sequência recebe o nome de *caminho simples* ou *elementar*. Se as arestas forem distintas, a sequência denomina-se *trajeto*. Um *ciclo* é um caminho v_1, \dots, v_k, v_{k+1} sendo $v_1 = v_{k+1}$ e $k \geq 3$. Se o caminho v_1, \dots, v_k for simples, o ciclo v_1, \dots, v_k, v_{k+1} também é denominado *simples* ou *elementar*. Um grafo que não possui ciclos simples é *acíclico*. Um *triângulo* é um ciclo de comprimento 3. Dois ciclos são considerados idênticos se um deles puder ser obtido do outro, através de uma permutação circular de seus vértices. Um caminho que contenha cada vértice do grafo exatamente uma vez é chamado *hamiltoniano*. Por outro lado, algum caminho ou ciclo que contenha cada aresta do grafo, também exatamente uma vez cada, é denominado *euleriano*. Um ciclo v_1, \dots, v_k, v_{k+1} é *hamiltoniano* quando o caminho v_1, \dots, v_k o for. Se G é um grafo que possui ciclo hamiltoniano ou euleriano, então G é denominado *hamiltoniano* ou *euleriano*, respectivamente. Para maior simplicidade de apresentação, quando não houver ambiguidade os termos “caminho” e “ciclo” serão utilizados com o significado de “caminho simples” e “ciclo simples”, respectivamente.

Um grafo $G(V, E)$ é denominado *conexo* quando existe caminho entre cada par de vértices de G . Caso contrário G é *desconexo*. Observe que a representação geométrica de um grafo desconexo é necessariamente descontígua. Em especial, o grafo G será *totalmente desconexo* quando não possuir arestas.

Seja S um conjunto e $S' \subseteq S$. Diz-se que S' é *maximal* em relação a uma certa propriedade P , quando S' satisfaz à propriedade P e não existe subconjunto $S' \subset S''$, que também satisfaz P . Observe que a definição acima não implica necessariamente em que S' seja o *maior* subconjunto de S satisfazendo P . Implica apenas no fato de que S' não está propriamente contido em nenhum subconjunto de S que satisfaça P . De maneira análoga, define-se também conjunto *minimal* em relação a uma certa propriedade. A noção de conjunto maximal (ou minimal) é frequentemente encontrada em combinatória. Ela pode ser aplicada, por exemplo, na seguinte definição. Denominam-se *componentes conexas* de um grafo G aos subgrafos maximais de G que sejam conexos. Observe que a propriedade P , nesse caso, é equivalente a “ser conexo”. As componentes conexas de um grafo G são pois os subgrafos de G correspondentes às porções contíguas de sua

representação geométrica.

Denomina-se *distância* $d(v,w)$ entre dois vértices v,w de um grafo ao comprimento do menor caminho entre v e w .

Seja $G(V,E)$ um grafo, $e \in E$ uma aresta. Denota-se por $G - e$ o grafo obtido de G , pela exclusão da aresta e . Se v,w é um par de vértices não adjacentes em G , a notação $G + (v,w)$ representa o grafo obtido adicionando-se a G a aresta (v,w) . Analogamente, seja $v \in V$ um vértice de G . O grafo $G - v$ denota aquele obtido de G pela remoção do vértice v . Observe que excluir um vértice implica em remover de G o vértice em questão e as arestas a ele incidentes. Da mesma forma, $G + w$ representa o grafo obtido adicionando-se a G o vértice w .

Observa-se ainda que essas operações de inclusão e exclusão de vértices e arestas podem ser generalizadas. De um modo geral, se G é um grafo e S um conjunto de arestas ou vértices, $G - S$ e $G + S$ denotam, respectivamente, o grafo obtido de G pela exclusão e inclusão de S , respectivamente.

Um grafo é *completo* quando existe uma aresta entre cada par de seus vértices. Utiliza-se a notação K_n , para designar um grafo completo com n vértices. O grafo K_n possui pois o número máximo possível de arestas para um dado n , ou seja $\binom{n}{2}$ arestas.

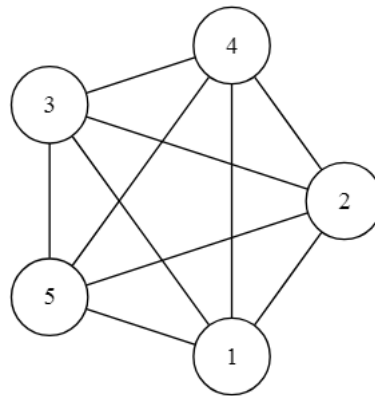


Figura 2 – O grafo K_5

Um grafo $G(V,E)$ é *bipartido* quando o seu conjunto de vértices V puder ser particionado em dois subconjuntos V_1, V_2 , tais que toda aresta de G une um vértice de V_1 a outro de V_2 . É útil denotar-se o grafo bipartido G por $(V_1 \cup V_2, E)$. Um grafo *bipartido completo* possui uma aresta para cada par de vértices v_1, v_2 , onde $v_1 \in V_1$ e $v_2 \in V_2$. Sendo $n_1 = |V_1|$ e $n_2 = |V_2|$, um grafo bipartido completo é denotado por K_{n_1, n_2} e obviamente possui $n_1 n_2$ arestas.

Um grafo S_k é chamado de *estrela* se S_k é o grafo bipartido completo $K_{1,k}$, $k \geq 1$. Também temos que $S_{p,q}$ é chamado de *estrela dupla* se puder se obtido de S_p e S_q , respectivamente considerando v_p e v_q os seus centros, temos que $V(S_{p,q}) = V(S_p) \cup V(S_q)$ e $E(S_{p,q}) = E(S_p) \cup E(S_q) \cup \{(v_p, v_q)\}$.

Observe que Figura 3a representa a estrela S_5 e as duas partições são $V_1 = \{1\}$ e $V_2 = \{2, 3, 4, 5\}$. Na Figura 3b, podemos considerar S_p com os vértices $\{1, 3, 4, 5, 6\}$ com o centro no vértice 1 e S_q com $\{2, 7, 8, 9\}$ com o centro no vértice em 2.

Um *subgrafo* $G_2(V_2, E_2)$ de um grafo $G_1(V_1, E_1)$ é um grafo tal que $V_2 \subseteq V_1$ e $E_2 \subseteq E_1$. Se além disso, G_2 possuir toda aresta (v,w) de G_1 tal que ambos v e w estejam em V_2 , então G_2 é o *subgrafo induzido pelo subconjunto de vértices* V_2 , denotado como $G \langle V_2 \rangle$. Diz-se então que V_2 induz G_2 . Ou seja, o subgrafo induzido G_2 de G_1 satisfaz: para

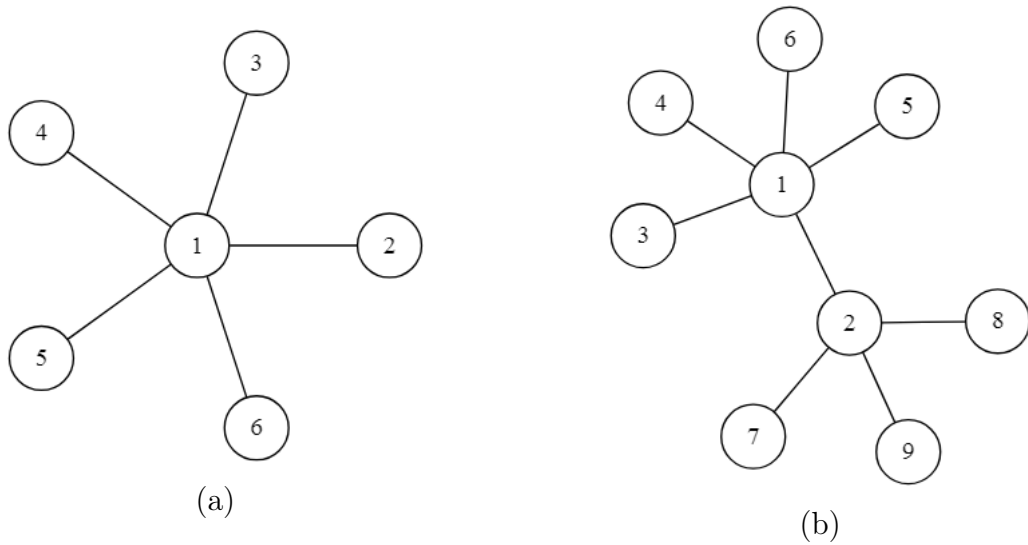


Figura 3 – A estrela S_5 e a estrela dupla $S_{4,3}$

$v, w \in V_2$, se $(v, w) \in E_1$ então $(v, w) \in E_2$. Similarmente, para um dado $E_2 \subseteq E_1$, se $V_2 \subseteq V_1$ for exatamente o subconjunto de vértices de V_1 que são incidentes a alguma aresta de E_2 , então $G_2 < V_2, E_2 >$ é o subgrafo induzido pelo subconjunto de arestas $E_2 \subseteq E_1$.

Denomina-se *clique* de um grafo G a um subgrafo de G que seja completo. Chama-se *conjunto independente de vértices* a um subgrafo induzido de G que seja totalmente desconexo. Ou seja, numa clique existe uma aresta entre cada par de vértices distintos. Num conjunto independente de vértices não há aresta entre qualquer par de vértices. O *tamanho* de uma clique ou conjunto independente de vértices é igual à cardinalidade de seu conjunto de vértices. Dado um grafo G , o problema de encontrar em G uma clique ou conjunto independente de vértices com um dado tamanho k , pode ser facilmente resolvido por um processo de força bruta, examinando o subgrafo induzido de cada um dos $\binom{n}{k}$ subconjuntos de V com k vértices. Este método corresponde pois a um algoritmo de complexidade $\Omega(n^k)$. É desconhecido se existe algum processo eficiente para resolver este problema para um k arbitrário.

Um grafo G é um *grafo split* se $V(G)$ pode ser particionado em dois conjuntos de vértices V_1 e V_2 cujos grafos induzidos sejam uma clique e um conjunto independente. Observe que um grafo completo é sempre um grafo split, já que o conjunto que representa a clique são todos os vértices do grafo e o conjunto independente é vazio.

Note que a Figura 4 apresenta um grafo split no qual o subgrafo induzido por $V_1 = \{1, 2, 3, 4, 5\}$ é uma clique e o subgrafo induzido por $V_2 = \{6, 7, 8, 9, 10\}$ é um conjunto independente.

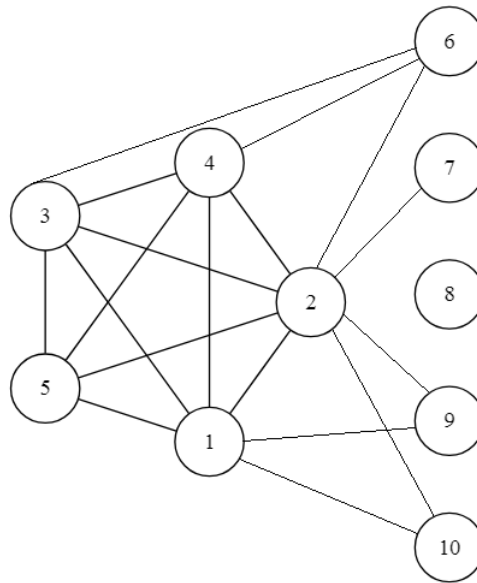


Figura 4 – Um grafo split

1.3 Árvores

Um grafo que não possui ciclos é acíclico. Denomina-se *árvore* a um grafo $T(V,E)$ que seja acíclico e conexo. Se um vértice v da árvore T possuir $\text{grau}(v) \leq 1$ então v é uma *folha*. Caso contrário, se $\text{grau}(v) > 1$ então v é um *vértice interior*. Um conjunto de árvores é denominado *floresta*. Assim sendo, todo grafo acíclico é uma floresta.

Observe que toda árvore T com n vértices possui exatamente $n - 1$ arestas. O seguinte argumento indutivo justifica a afirmativa. Se T possui 1 vértice então obviamente seu número de arestas é zero. Suponha que T contenha $n - 1$ vértices e $n - 2$ arestas, $n > 1$. Considere a adição de uma nova aresta $e = (v,w)$. Como T é conexo, pelo menos um dentre v,w pertence a T . Mas como T é acíclico, pelo menos um deles não pertence a T (caso contrário, a inclusão de uma nova aresta entre dois vértices de T produz um ciclo). Logo, exatamente um dentre v,w pertence a T , o que significa que a árvore passa a possuir n vértices e $n - 1$ arestas. Através de um raciocínio análogo pode-se mostrar que o número de folhas de T varia entre um mínimo de 2 e máximo de $n - 1$, para $n > 2$.

Árvores constituem uma classe extremamente importante de grafos, principalmente devido a sua aplicação nas mais diversas áreas. O teorema abaixo é uma caracterização para as árvores.

Teorema 1. Um grafo G é uma árvore se e somente se existir um único caminho entre cada par de vértices de G .

Demonstração. Seja G uma árvore. Então G é conexo e portanto existe pelo menos um caminho entre cada par v,w de vértices de G . Suponha que existam dois caminhos distintos vP_1w e vP_2w , entre v e w . Então vP_1wP_2v é um ciclo, o que contradiz G ser acíclico. Isto prova a necessidade. Reciprocamente, se existe exatamente um caminho entre cada par de vértices de G , então o grafo é conexo e, além disso, não pode conter ciclos. Logo G é uma árvore. \square

Além do teorema acima, existem diversas outras possíveis caracterizações para as árvores. O teorema seguinte resume algumas dessas.

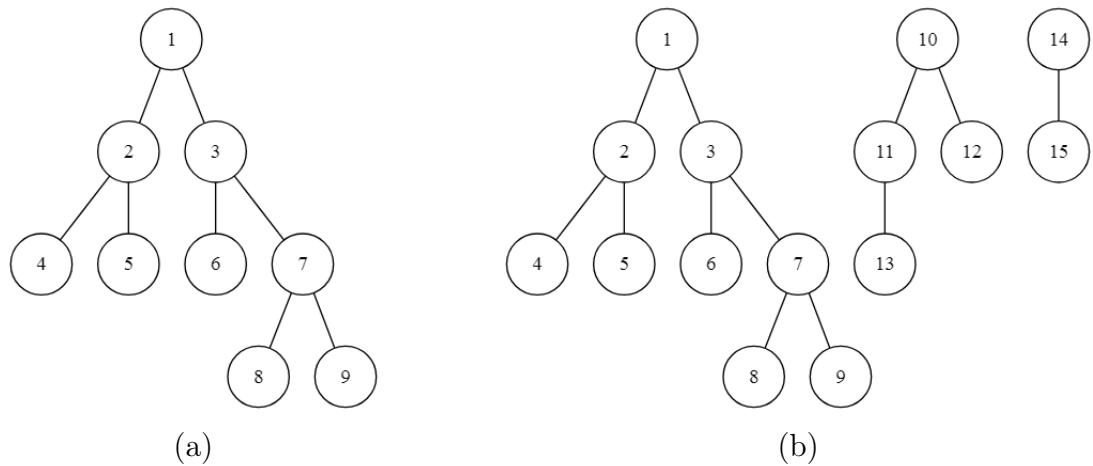


Figura 5 – Uma árvore e uma floresta

Teorema 2. Seja $G(V,E)$ um grafo. As possíveis afirmativas são equivalentes.

- G é uma árvore.
- G é conexo e $|E|$ é mínimo.
- G é conexo e $|E| = |V| - 1$.
- G é acíclico e $|E| = |V| - 1$.
- G é acíclico e para todo $v, w \in V$, a adição da aresta (v, w) produz um grafo contendo exatamente um ciclo.

Uma árvore $T(V,E)$ é denominada *enraizada* quando algum vértice $r \in V$ é escolhido como especial. Esse vértice r é então chamado de *raiz* da árvore e denotaremos essa árvore por T^r . Uma árvore não enraizada é também denominada *árvore livre*. Por exemplo, a árvore livre da (a) torna-se a árvore enraizada da (b), quando o vértice c é escolhido como raiz. Sejam v, w dois vértices da árvore T^r . Suponha que v pertença ao caminho de r a w em T . Então v é *ancestral* de w , sendo w *descendente* de v . Se ainda $v \neq w$, v é *ancestral próprio* de w , e este é *descendente próprio* de v . Além disso, se (v, w) é aresta de T , então v é *pai* de w , sendo w *filho* de v . Dois vértices que possuem o mesmo pai são *irmãos*.

A raiz de uma árvore naturalmente não possui pai, enquanto que todo vértice $v \neq r$ possui um único. Uma *folha* é um vértice que não possui filhos. Denomina-se *nível* de um vértice v , denotado por $\text{nível}(v)$, ao número de vértices do caminho da raiz r a v . Então $\text{nível}(r) = 1$ e se dois vértices v, w são irmãos, $\text{nível}(v) = \text{nível}(w)$. A *altura* da árvore T é igual ao valor máximo de $\text{nível}(v)$, para todo vértice v de T .

Sejam $T^r(V,E)$, $r \in V$ e $v \in V$. Uma *subárvore* T_v^r de T^r é a árvore enraizada cuja raiz é v , definida pelo subgrafo induzido em T^r pelos descendentes de v . Isto é, u é pai de w em T_v^r se e somente se o for em T^r , para todo u, w descendentes de v em T^r . Seja S um subconjunto de vértices de T_v^r tal que $T_v^r - S$ seja conexo e $v \notin S$. O grafo $T_v^r - S$ é então denominado *subárvore parcial* de raiz v . Obviamente a subárvore de raiz v é única para cada $v \in V$, enquanto que a subárvore parcial não o é.

Observe que na definição da árvore enraizada é irrelevante a ordem em que os filhos de cada vértice v são considerados. Caso esta ordenação seja relevante, a árvore é denominada *enraizada ordenada*. Assim sendo, numa árvore enraizada ordenada, para cada

vértice v que possui filhos, pode-se identificar o 1 filho de v (o mais à esquerda), o 2 (segundo mais à esquerda) e assim por diante. Obviamente, se duas árvores enraizadas são isomorfas, não necessariamente elas o serão como árvores enraizadas ordenadas. Para tanto, o isomorfismo deve preservar também a ordenação. Observe, por outro lado, que árvores isomorfas livres também não serão necessariamente isomorfas, se consideradas como enraizadas. Há diversas aplicações em árvores, que requerem a mencionada ordenação.

Uma *árvore estritamente m -ária* T é uma árvore enraizada ordenada em que cada vértice não folha possui exatamente m filhos, $m \geq 1$. Quando $m = 2$ a árvore é *estritamente binária*. A cada filho w de todo vértice não folha v de T , atribua agora um rótulo inteiro positivo $r(w)$, $1 \leq r(w) \leq m$, igual à posição que w ocupa na ordenação dos filhos de v . Uma subárvore parcial de uma árvore estritamente m -ária que possui essa rotulação é chamada *árvore m -ária*. Ou seja, numa árvore m -ária T dois vértices irmãos w_j, w_{j+1} consecutivos na ordenação dos filhos de um vértice v podem ter rótulos não consecutivos (não necessariamente $r(w_{j+1}) - r(w_j) = 1$). Pode-se então considerar que uma árvore m -ária é obtida a partir de uma estritamente m -ária pela remoção de $r(w_{j+1}) - r(w_j) - 1$ subárvores entre cada par de irmãos w_j, w_{j+1} , além de $r(w_1) - 1$ subárvores antes do filho w_1 de v e $r(w_f) - 1$ após o último filho w_f de v .

Numa árvore binária cada filho w de v pode ser identificado como o *filho esquerdo* ou *direito*. Naturalmente, o filho esquerdo pode existir sem o direito, ou vice-versa.

1.4 Conectividade

Segundo (BLAIR; PEYTON, 1991), um subconjunto $S \subset V(G)$ é dito um *separador* de G se existem dois vértices que estão na mesma componente conexa de G e em componentes conexas distintas de $G - S$. Se a e b são dois vértices que podem ser separados por S , então S é dito também como *$a - b$ separador*. Também dizemos que S é um *separador minimal* de G se S é um separador e não há subconjunto próprio de S que também seja um separador. Da mesma forma, S é um *$a - b$ separador minimal* se S é um $a - b$ separador e não há subconjunto próprio de S que também seja um $a - b$ separador. Podemos não especificar o par de vértices do separador, referindo-nos a S simplesmente como um *separador de vértices minimal*. Observamos que todo separador minimal é um separador de vértices minimal para algum par de vértices, mas não o contrário.

Observe a Figura 6 que mostra um grafo com um único separador minimal $\{4\}$ e dois separadores de vértices minimais $\{4\}$ e $\{1, 4\}$.

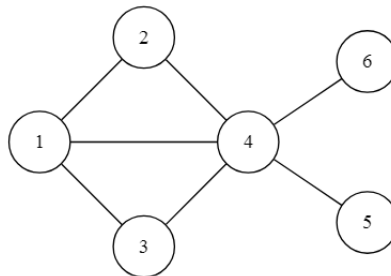


Figura 6 – Um grafo com um único separador minimal e dois separadores de vértices minimais

Denomina-se *conectividade de vértices* c_V de G à cardinalidade do menor separador de G ou $n - 1$ caso G não possua separador. Sendo k um inteiro positivo, diz-se que um

grafo G é k -conexo em vértices quando a sua conectividade de vértices é $\geq k$. Observe que se G for o grafo completo K_n , então $c_V = n - 1$, ou seja, não existe subconjunto próprio de vértices $V' \subset V$ cuja remoção desconecte G , mas removendo-se $n - 1$ vértices resulta o grafo trivial.

Seja $G(V, E)$ um grafo conexo. Um *corte de arestas* de G é um subconjunto minimal de arestas $E' \subseteq E$, cuja remoção de G o desconecta. Isto é, $G - E'$ é desconexo e, para todo subconjunto próprio $E'' \subset E'$, $G - E''$ é conexo.

Considere $G(V, E)$ um grafo. Um vértice v é denominado *articulação* quando sua remoção de G o desconecta. Ou seja, $G - v$ é desconexo. Uma aresta $e \in E$ é chamada *ponte* quando sua remoção de G o desconecta. Nesse caso, $G - e$ é desconexo. Assim sendo, um grafo é biconexo em vértices (arestas) se e somente se não possuir articulações (pontes). Denominam-se *componentes biconexas* do grafo G aos subgrafos maximais de G que sejam biconexos em vértices, ou isomorfos a K_2 . Cada componente biconexa é também chamada *bloco* do grafo. Assim, se G é biconexo em vértices, então G possui um único bloco, que coincide com o próprio G .

O lema seguinte traz informações adicionais sobre os blocos de um grafo.

Lema 1. Seja G um grafo. Então:

1. Cada aresta de G pertence a exatamente um bloco do grafo.
2. Um vértice v de G é articulação se e somente se v pertencer a mais de um bloco do grafo.

A prova é simples e será omitida. As articulações e pontes de um grafo podem ser caracterizadas pelo lema a seguir.

Lema 2. Seja $G(V, E)$ um grafo conexo, $|V| > 2$. Então:

1. Um vértice $v \in V$ é articulação de G se e somente se existirem vértices $w, u \neq v$ tais que v está contido em todo caminho entre w e u em G .
2. Uma aresta $(p, q) \in E$ é ponte se e somente se p, q for o único caminho simples entre p e q em G .

Demonstração.

Se v é articulação de G então $G - v$ é desconexo. Sejam w, u dois vértices localizados em componentes conexas distintas de $G - v$. Então todo caminho entre w e u contém v . Analogamente segue a recíproca.

Se $(p, q) \in E$ é uma ponte então o grafo $G - (p, q)$ é desconexo, localizando-se p e q em componentes conexas distintas de $G - (p, q)$. Logo (p, q) é o único caminho simples entre p e q em G ((b)). Da mesma forma, a recíproca é análoga. \square

Lema 3. Um grafo $G(V, E)$, $|V| > 2$, é biconexo se e somente se cada par de vértices de G está contido em algum ciclo.

Demonstração. Segue da parte (i) do lema anterior. \square

O lema anterior pode ser generalizado, como se segue.

Teorema 3. Seja G um grafo k -conexo. Então existe algum ciclo de G passando por cada subconjunto de k vértices.

Observe que a recíproca do teorema acima não é válida para $k > 2$. O grafo definido por exatamente um ciclo de comprimento $|V|$ explica a razão.

O teorema seguinte caracteriza grafos k -conexos.

Teorema 4. Um grafo $G(V,E)$ é k -conexo, se e somente se existem k caminhos disjuntos (exceto nos extremos) entre cada par de vértices de G .

As provas dos Teoremas anteriores são mais elaboradas e não serão aqui apresentadas.

1.5 Grafos Bloco

O *grafo bloco*, também chamado de *árvore de clique* (VUŠKOVIĆ, 2010), é um tipo de grafo não direcionado em que toda componente biconexa é uma clique. Vale destacar que a classe árvores é uma subclasse de grafos bloco.

Os grafos bloco são exatamente os grafos para os quais, para cada quatro vértices u, v, x e y , as duas maiores das três distâncias $d(u,v) + d(x,y)$, $d(u,x) + d(v,y)$ e $d(u,y) + d(v,x)$ são sempre iguais (HOWORKA, 1979)(BANDELT; MULDER, 1986).

Os grafos bloco também têm uma caracterização de grafo por não terem o grafo de diamante ou um ciclo de quatro ou mais vértices como um subgrafo induzido; isto é, são os grafos cordais sem diamante (BANDELT; MULDER, 1986). Eles são também os grafos ptolemaicos (grafos de distância hereditária cordais) nos quais para todo par de vértices com distância 2, vale que esses vértices estão conectados por um único menor caminho (HOWORKA, 1979), e os grafos cordais em que a cada duas cliques maximais têm no máximo um vértice comum (HOWORKA, 1979).

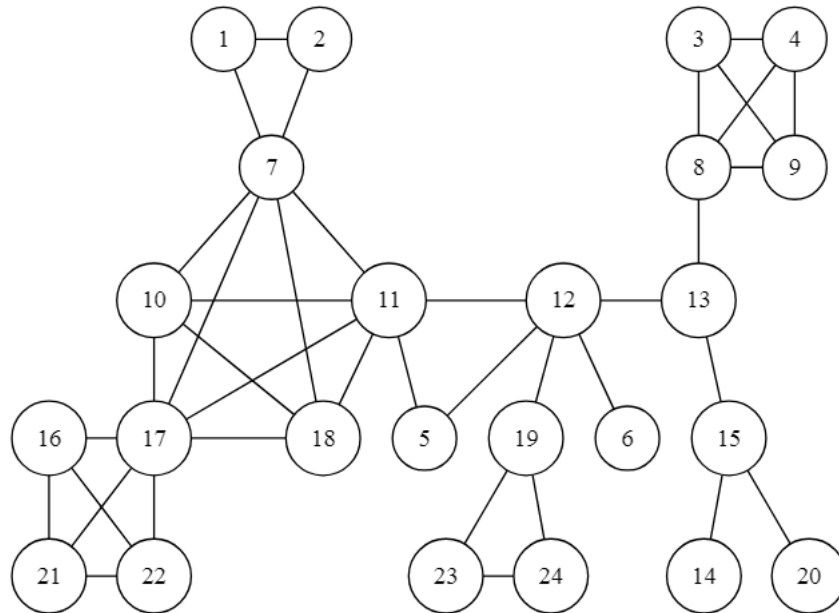


Figura 7 – Um grafo bloco

1.6 Grafos Cordais

Seja $G(V,E)$ um grafo não direcionado e C um ciclo de G . Uma *corda* de G é uma aresta não pertencente a C que incidente a dois vértices de C . Como exemplo, observamos no grafo da Figura 8a que a aresta $(2,4)$ é uma corda do ciclo $\{1, 2, 5, 4, 1\}$.

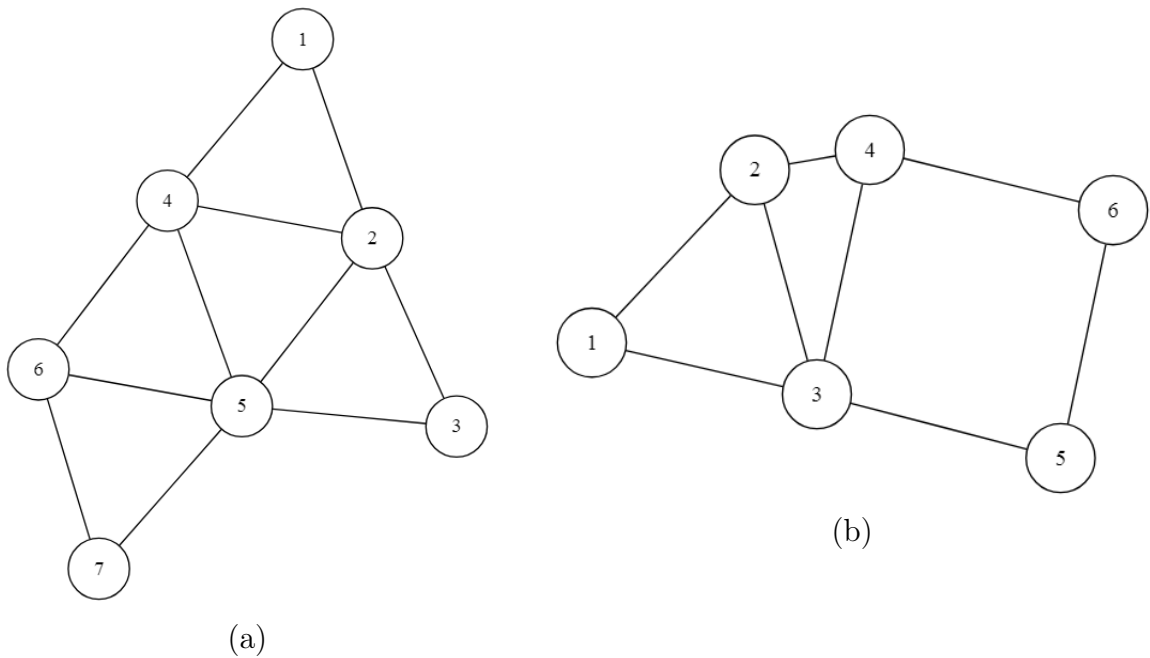


Figura 8 – Um grafo cordal e um grafo não cordal

Um grafo $G(V,E)$ é denominado *cordal* (ou *triangularizado*) se todo ciclo de G com comprimento maior do que 3 possui uma corda. Como exemplo, observamos o grafo da Figura 8a é cordal, enquanto que o da Figura 8b não é, pois o ciclo $\{3, 4, 6, 5, 3\}$ de comprimento 4 não possui corda.

Um vértice $v \in V$ é denominado *simplicial* se o subgrafo induzido pelos vértices adjacentes de v for completo. Por exemplo, os vértices 1, 3 e 7 do grafo da Figura 8a são simpliciais.

Algoritmos de reconhecimento de um grafo cordal podem ser feitos em tempo linear conforme (ROSE; TARJAN; LUEKER, 1976) (TARJAN; YANNAKAKIS, 1984) (BRANDSTÄDT; LE; SPINRAD, 1999).

Teorema 5. Um grafo G é cordal se e somente se todo separador minimal de vértices de G é completo em G (DIRAC, 1961).

Demonstração. Assuma que todo separador minimal de vértices de G é completo em G e que $\mu = v_0, \dots, v_k, v_0$ seja um ciclo qualquer de comprimento maior que 3 em G , ou seja, $k \geq 3$. Se $(v_0, v_2) \in E$, então μ tem uma corda. Senão, então existe um $v_0 - v_2$ separador S (por exemplo, $S = V - \{v_0, v_2\}$). Além disso, qualquer separador desse tipo tem que conter v_1 e v_i para algum i , $3 \leq i \leq k$. Escolha um S para ser um $v_0 - v_2$ separador tal que S , por hipótese, é completo em G . Então (v_1, v_i) é uma corda de μ , o que prova a primeira parte do Teorema.

Agora assumamos que G seja cordal e que S seja um $a - b$ separador minimal de G . Seja $G(A)$ e $G(B)$ as componentes conexas de $G - S$ contendo a e b , respectivamente. Isso é suficiente para mostrar que para qualquer dois vértices distintos em S , digamos x e y , temos $(x, y) \in E$. Como S é minimal, todo vértice $v \in S$ é adjacente a algum vértice em A e algum vértice em B , caso contrário $S - \{v\}$ seria um $a - b$ separador contrariando que S é minimal. Portanto, existem caminhos $\mu = \{x, a_1, \dots, a_r, y\}$ e $v = \{y, b_1, \dots, b_t, x\}$ tais que todo $a_i \in A$ e todo $b_i \in B$. Além disso, escolha μ e v tais que eles tenham menor comprimento possível maior que o outro e os combine para formar um ciclo $\sigma =$

$\{x, a_1, \dots, a_r, y, b_1, \dots, b_t, x\}$. Como G é cordal e σ é um ciclo de comprimento maior que 3, σ tem que possuir uma corda. Qualquer corda de σ incidente a a_i , $1 \leq i \leq r$ ou une a_i a um outro vértice em μ , contrariando que r é minimal, ou une a_i a algum vértice de B , o que é impossível pois S separa A de B em G . Consequentemente, nenhuma corda de σ é incidente a um vértice a_i , $1 \leq i \leq r$ e, pelo mesmo argumento, nenhuma corda do ciclo é incidente a um vértice b_j , $1 \leq j \leq t$. Portanto, a única corda possível é (x, y) . \square

Para qualquer grafo cordal G , há diversas representações, sendo uma delas árvores chamadas de *clique tree*. Essas árvores se mostram como uma maneira compacta e eficiente de representar grafos cordais (TARJAN; YANNAKAKIS, 1984) (LEWIS; PEYTON; POTHEN, 1989). Um grafo cordal sempre pode ser representado por uma ou mais clique trees. Conforme (BLAIR; PEYTON, 1991) (IBARRA, 2009), definimos T como uma clique tree do grafo G se os vértices de T correspondem às cliques maximais de G e T possui a *propriedade de interseção de cliques*. Essa propriedade determina que, para quaisquer duas cliques maximais K e K' de G , o conjunto $K \cap K'$ está contido em todas as cliques maximais no caminho de K até K' em T . Também é provado em (BLAIR; PEYTON, 1991) que as arestas (K, K') de T correspondem a todos os separadores de vértices minimais $K \cap K'$ de G .

Portanto, para qualquer clique tree T de um grafo cordal G , os vértices e as arestas de T correspondem às cliques maximais e aos separadores de vértices minimais de G , respectivamente. Uma clique maximal de G corresponde a exatamente um vértice de T enquanto um separador de vértices minimal de G corresponde a pelo menos uma aresta de T . Desse modo, como um grafo cordal G possui no máximo n cliques maximais (FULKERSON; GROSS, 1965), então G possui no máximo $n - 1$ separadores de vértices minimais.

Algoritmos para a geração de uma clique tree de um grafo cordal conexo podem ser feitos em tempo linear, conforme descrito em (BLAIR; PEYTON, 1991) (BRANDSTÄDT; LE; SPINRAD, 1999) (SPINRAD, 2003) (BERRY; SIMONET, 2016).

Observe um grafo cordal G na Figura 9 e uma clique tree T correspondente na Figura 10. As cliques maximais de G são vértices de T , como, por exemplo, o vértice c de T que corresponde à clique $\{4, 5, 12, 13\}$ de G e o vértice i de T corresponde à clique $\{14, 15, 16\}$ de G . Observe também que os separadores de vértices minimais de G correspondem às arestas de T , como por exemplo o separador $\{14, 15\}$ em G que é a aresta (f, i) em T e o separador $\{4, 5\}$ em G que é a aresta (b, c) em T .

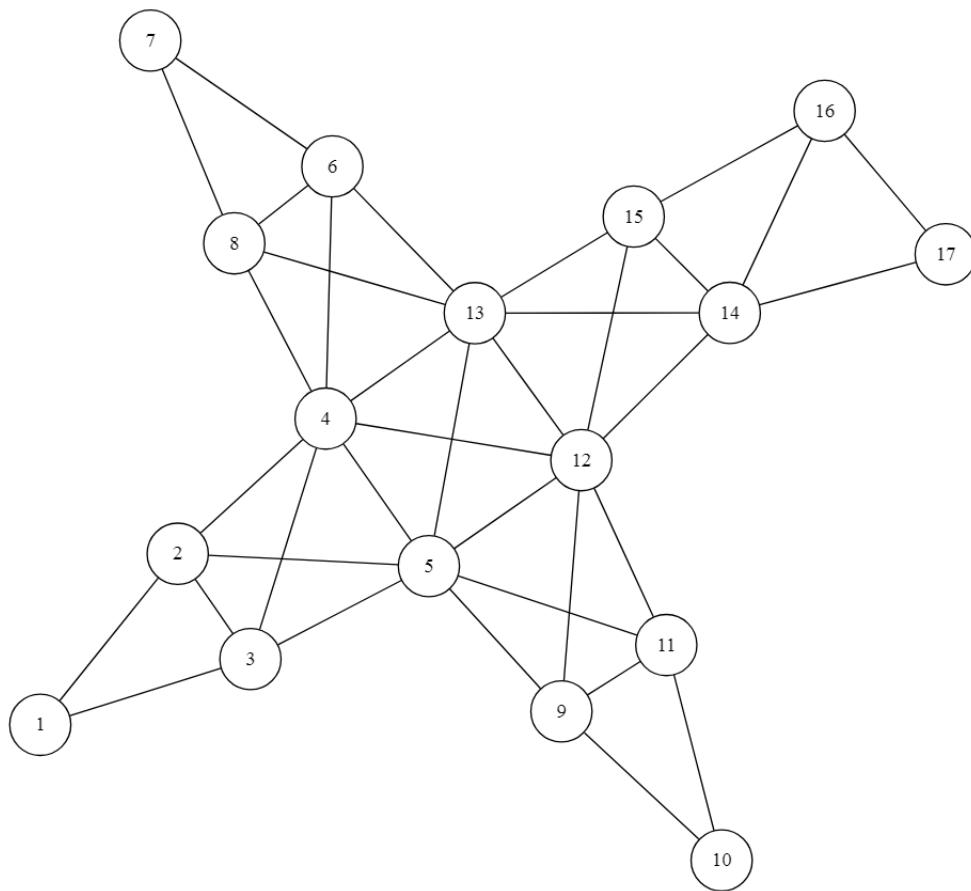


Figura 9 – Um grafo cordal

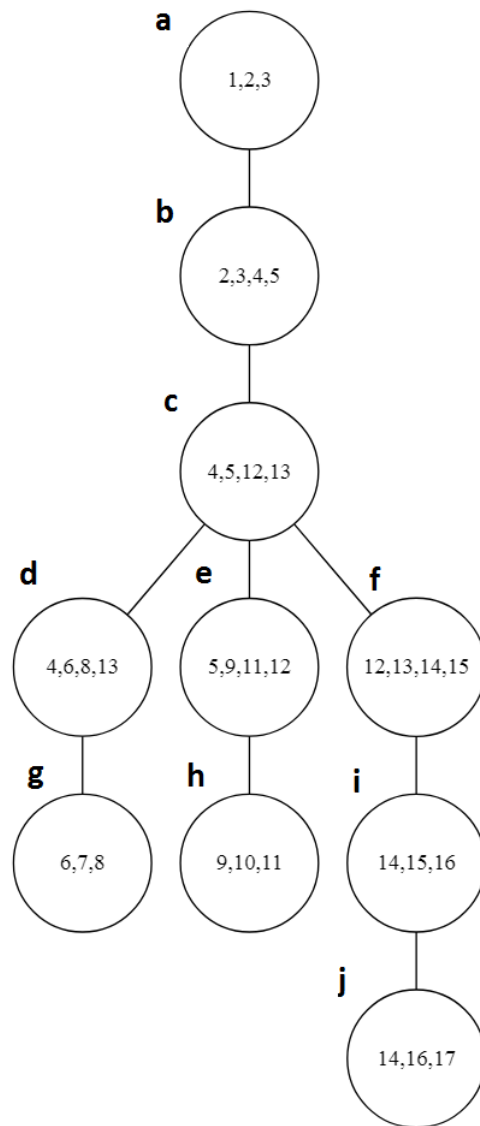


Figura 10 – Uma clique tree correspondente ao grafo cordal da Figura 9

1.7 Grafos Direcionados

Os grafos examinados até o momento são também denominados *não direcionados*. Isto porque suas arestas (v,w) não são direcionadas. Assim, se (v,w) é aresta de $G(V,E)$ então tanto v é adjacente a w como w é adjacente a v . Em contrapartida, um *grafo direcionado* (*digrafo*) $D(V,E)$ é um conjunto finito não vazio V (*os vértices*), e um conjunto E (*as arestas*) de pares ordenados de vértices distintos. Portanto, num digrafo cada aresta (v,w) possui uma única direção de v para w . Diz-se também que (v,w) é *divergente* de v e *convergente* a w . Boa parte da nomenclatura e dos conceitos é análoga nos dois casos. Assim sendo, definem-se, por exemplo, caminho simples, trajeto, ciclo e ciclo simples de forma análoga às definições de grafos. Em particular, utilizamos o termo *subdigrafo* com o significado de subgrafo direcionado. Contudo, ao contrário dos grafos, os digrafos podem possuir ciclos de comprimento 2, no caso em que ambos (v,w) e (w,v) são arestas do digrafo. Os caminhos e ciclos em um digrafo devem obedecer ao direcionamento das arestas.

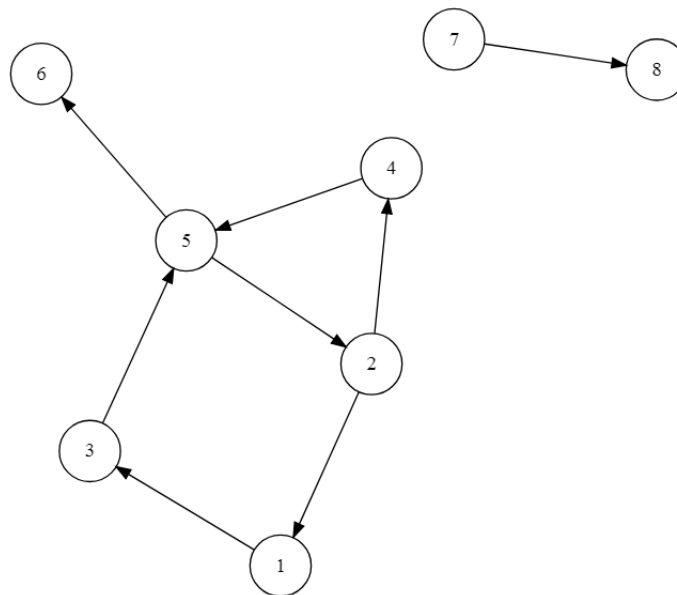


Figura 11 – Um digrafo

Seja $D(V,E)$ um digrafo e um vértice $v \in V$. O *grau de entrada* de v é o número de arestas convergentes a v . O *grau de saída* de v é o número de arestas divergentes de v . Uma *fonte* é um vértice com grau de entrada nulo, enquanto que um *sumidouro* (ou *poço*) é um com grau de saída nulo.

Se forem retiradas as direções das arestas de um digrafo D obtém-se um multigrafo não direcionado, chamado *grafo subjacente* a D .

Um digrafo $D(V,E)$ é *fortemente conexo* quando para todo par de vértices $v,w \in V$ existir um caminho em D de v para w , e também de w para v . Se ao menos um desses caminhos existir para todo $v,w \in V$, então D é *unilateralmente conexo*. Um digrafo é chamado *fracamente conexo* ou *desconexo*, conforme seu grafo subjacente seja conexo ou desconexo, respectivamente. Observe que se um digrafo é fortemente conexo então também é unilateralmente e fracamente conexo. Uma *componente fortemente conexa* é um subdigrafo maximal de D , que seja fortemente conexo. Se existir algum caminho em D de um vértice v para um vértice w , então diz-se que v *alcança* ou *atinge* w , sendo este

alcançável ou *atingível* de v . Se v alcançar todos os vértices de D então v é chamado *raiz* do digrafo.

Um digrafo é *acíclico* quando não possui ciclos. Observe que o grafo subjacente a um digrafo acíclico não é necessariamente acíclico. Mas é acíclico um digrafo cujo grafo subjacente também o seja.

Finalmente, uma *árvore direcionada enraizada* é um digrafo D tal que exatamente um vértice r possui grau de entrada nulo, enquanto para todos os demais vértices o grau de entrada é igual a um. É imediato observar que D é acíclico com exatamente uma raiz r . Além disso, o grafo subjacente T de D é uma árvore. Por isso é usual aplicar a nomenclatura de árvores a esta classe de digrafos.

1.8 Representação de Grafos

Nesta seção examinam-se algumas representações de grafos, adequadas ao uso em computador. Ou seja, estruturas que (i) correspondam univocamente a um grafo dado e (ii) possam ser armazenadas e manipuladas sem dificuldade, em um computador. Observe que a representação geométrica, por exemplo, não satisfaz à condição (ii). Dentre os vários tipos de representações adequadas ao computador, ressaltam-se as *representações matriciais* e as *por listas*. A seguir, são examinadas algumas dessas.

Dado um grafo $G(V, E)$ a *matriz de adjacências* $R = (r_{ij})$ é uma matriz binária $n \times n$ tal que:

$$r_{ij} = 1 \Leftrightarrow (v_i, v_j) \in E$$

Ou seja, $r_{ij} = 1$ quando os vértices v_i, v_j forem adjacentes e $r_{ij} = 0$, caso contrário. É imediato verificar que a matriz de adjacências representa um grafo sem ambiguidade. Além disso, é de simples manipulação em computador. Algumas propriedades da matriz R são imediatas. Por exemplo, R é simétrica para um grafo não direcionado. Além disso, o número de 1's é igual a $2m$, pois cada aresta (v_i, v_j) dá origem a dois 1's em A , r_{ij} e r_{ji} .

Observe que uma matriz de adjacências caracteriza univocamente um grafo. Contudo a um mesmo grafo G podem corresponder várias matrizes diferentes. De fato, se R_1 é matriz de adjacências de G e R_2 é outra matriz obtida por alguma permutação de linhas e colunas de R_1 , então R_2 também é matriz de adjacências de G . Ou seja, para construir uma matriz de adjacências de $G(V, E)$ é necessário arbitrar uma certa permutação v_1, v_2, \dots, v_n para os vértices de V . Naturalmente, permutações diferentes podem conduzir a matrizes diferentes. Observe que o problema de isomorfismo de grafos pode ser então formulado nos seguintes termos: “sendo R_1 e R_2 duas matrizes de adjacências dadas, representam R_1 e R_2 o mesmo grafo?”. As Figuras (a) e (b) ilustram duas matrizes de adjacências do grafo da , correspondentes às permutações dos vértices 1,2,3,4,5,6 e 5,4,1,3,2,6, respectivamente.

A matriz de adjacências pode ser também definida para digrafos. Basta tomar $r_{ij} =$

$$\begin{array}{cc} \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \end{pmatrix} \\ \text{(a)} & \text{(b)} \end{array}$$

Figura 12 – Matrizes de adjacências

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Figura 13 – Matriz de incidências

$1 \Leftrightarrow (v_i, v_j)$ for aresta divergente de v_i e convergente a v_j , e $r_{ij} = 0$, caso contrário. Naturalmente, a matriz não é mais necessariamente simétrica e o número de 1's é exatamente igual a m .

Em relação ao espaço necessário ao armazenamento da matriz, em qualquer caso existem n^2 informações binárias, o que significa um espaço $O(n^2)$. Ou seja, um espaço não linear com o tamanho do grafo (número de vértices e arestas), no caso em que este for esparsos (isto é, $m = O(n)$). Esta é uma das principais desvantagens dessa representação.

Uma outra representação matricial para o grafo $G(V, E)$ é a *matriz de incidências* $n \times m$ de $B = (b_{ij})$ assim definida:

$b_{ij} = 1 \Leftrightarrow$ vértice v_i e aresta e_j forem incidentes

$b_{ij} = 0$ caso contrário.

Ou seja, arbitram-se permutações para os vértices v_1, \dots, v_n e para as arestas e_1, \dots, e_m de G . Então $b_{ij} = 1$ precisamente quando o vértice v_i for um extremo da aresta e_j , e $b_{ij} = 0$, caso contrário. Também nesse caso é imediato verificar que a matriz de incidências representa univocamente um grafo, mas este último pode ser representado, em geral, por várias matrizes de incidências diferentes. Observe que cada coluna de B tem exatamente dois 1's. A ilustra a matriz de incidências do grafo da , correspondente às permutações de vértices 1, 2, 3, 4, 5, 6 e de arestas (1, 2), (1, 3), (1, 6), (1, 5), (2, 3), (2, 6), (3, 5), (3, 6), (5, 4), (5, 6), (4, 6), (3, 4). A complexidade de espaço da matriz de incidências é $O(nm)$, maior ainda do que a da matriz de adjacências.

Existem várias representações de grafos por listas. Dentre essas, é muito comum a *estrutura de adjacências*. Seja $G(V, E)$ um grafo. A estrutura de *adjacências*, A de G é um conjunto de n listas $A(v)$, uma para cada $v \in V$. Cada lista $A(v)$ é denominada *lista de adjacências* do vértice v , e contém os vértices w adjacentes a v em G . Ou seja, $A(v) = \{w | (v, w) \in E\}$.

Observe que cada aresta (v, w) dá origem a duas entradas na estrutura de adjacências, correspondentes a $v \in A(w)$ e $w \in A(v)$. Logo, a estrutura A consiste de n listas com um total de $2m$ elementos. A cada elemento w de uma lista de adjacências $A(v)$ associa-se um ponteiro, o qual informa o próximo elemento, se houver, após w em $A(v)$. Além disso, é necessária também a utilização de um vetor p , de tamanho n , tal que $p(v)$ indique o ponteiro inicial da lista $A(v)$. Assim sendo, se $A(v)$ for vazia, $p(v) = \emptyset$. Pode-se concluir então que o espaço utilizado por uma estrutura da adjacências é $O(n + m)$, ou seja, linear com o tamanho de G . Esta é uma das razões pelas quais a estrutura de adjacências talvez seja a representação mais comumente encontrada nas implementações dos algoritmos em grafos.

A estrutura da adjacência pode ser interpretada como uma matriz de adjacências representada sob a forma de matriz esparsa, isto é, na qual os zeros não estão presentes. Nesse caso, $v_j \in A(v_i)$ corresponderia a afirmar que $r_{ij} = 1$ na matriz de adjacências. Finalmente observe que a estrutura da adjacência pode ser definida para digrafos de

forma análoga. Isto é, para um digrafo $D(V, E)$ define-se uma lista de adjacências $A(v)$, para cada $v \in V$. A lista $A(v)$ é formada pelos vértices divergentes de v , isto é, $A(v) = \{w | (v, w) \in E\}$. A estrutura de adjacências, nesse caso, contém m elementos. É imediato verificar que o espaço requerido pela estrutura A para representar o digrafo D é também linear com o tamanho de D .

1.9 Buscas em grafos

Dentre as técnicas existentes para a solução de problemas algorítmicos em grafos, a busca ocupa lugar de destaque, pelo grande número de problemas que podem ser resolvidos através da sua utilização. Provavelmente, a importância dessa técnica é ainda maior quando o universo das aplicações for restrito aos algoritmos considerados eficientes. A busca visa resolver um problema básico, qual seja o de como explorar um grafo. Isto é, dado um grafo, deseja-se obter um processo sistemático de como caminhar pelos vértices e arestas do mesmo.

De imediato, surge uma dificuldade: não há um referencial geral para definir um percurso sistemático no grafo, de modo que fique determinado o próximo vértice a ser visitado na sequência de visitas. Em particular, como caminhar no grafo, de modo a visitar todos os vértices e arestas, evitando contudo repetições desnecessárias de visitas a um mesmo vértice ou aresta?

Esta última questão é de enorme dificuldade, de um modo geral, a não ser que sejam utilizados recursos adicionais, durante o caminhar, que permitam reconhecer se um dado vértice ou aresta já foi ou não visitado anteriormente. Comumente, esses recursos adicionais correspondem a um conjunto de até n *marcas*. Uma marca é associada a um vértice v com a finalidade de registrar que v foi visitado. Isso permite distingui-la dos vértices não visitados.

1.9.1 Busca em profundidade

Uma busca é dita em *profundidade* quando o critério de escolha de vértice marcado (a partir do qual será realizada a próxima exploração de aresta) obedecer a: “dentre todos os vértices marcados e incidentes a alguma aresta ainda não explorada, escolher aquele mais recentemente alcançado na busca”. Observe que a escolha de vértice marcado torna-se única e sem ambiguidade, segundo o critério apresentado. O seguinte algoritmo recursivo implementa esse processo.

 Algoritmo 1 – Busca em profundidade (recursivo)

```

1: função BUSCAEMPROFUNDIDADE(Grafo conexo:  $G(V, E)$ )
2:   para  $u$  vértice de  $G$  faça
3:     DESMARCA( $u$ )
4:    $v \leftarrow$  Vértice qualquer de  $G$ 
5:   BUSCA( $v, \emptyset$ )
6: função BUSCARECURSAO(Vértice:  $v$ , Vértice:  $u$ )
7:   MARCA( $v$ )
8:   para  $w$  vizinho de  $v$  faça
9:     se  $w$  não está marcado então
10:      visitar aresta  $(v, w)$ 
11:      BUSCARECURSAO( $w, v$ )
12:   senão
13:     visitar aresta de retorno  $(v, w)$ 

```

1.9.2 Busca em largura

Um outro critério para explorar o grafo, descrito na presente seção, corresponde à busca em *largura*. Uma busca é dita em largura quando o critério de escolha de vértice marcado obedecer a: “dentre todos os vértices marcados e incidentes a alguma aresta ainda não explorada, escolher aquele menos recentemente alcançado na busca”. Assim como a busca em profundidade pode ser implementada com o auxílio de uma pilha, a busca em largura utiliza uma fila. De fato, pode-se formular para esses dois casos algoritmos que sejam absolutamente idênticos, a menos da estrutura auxiliar utilizada (pilha ou fila). O algoritmo seguinte implementa essa busca.

 Algoritmo 2 – Busca em largura

```

1: função BUSCAEMLARGURA(Grafo conexo:  $G(V, E)$ )
2:   para  $u$  vértice de  $G$  faça
3:     DESMARCA( $u$ )
4:    $v \leftarrow$  Vértice qualquer de  $G$ 
5:    $Q \leftarrow$  Fila vazia
6:   MARCA( $v$ )
7:   inserir  $v$  em  $Q$ 
8:   enquanto  $Q \neq \emptyset$  faça
9:     seja  $v$  o primeiro elemento de  $q$ 
10:    para  $w$  vizinho de  $v$  faça
11:      se  $w$  não está marcado então
12:        visitar aresta  $(v, w)$ 
13:        MARCA( $w$ )
14:        inserir  $w$  em  $Q$ 
15:    senão
16:      se  $w \in Q$  então
17:        visitar aresta  $(v, w)$ 
18:    retirar  $v$  de  $Q$ 

```

2 EMPARELHAMENTOS

Neste capítulo, serão descritos alguns conceitos teóricos de emparelhamentos em grafos, usando as mesmas referências do Capítulo 1, ou seja, os livros (SZWARCFITER; MARKENZON, 2010; SZWARCFITER, 2018).

Seja $G(V, E)$ um grafo não direcionado, $|V| = n$ e $|E| = m$. Um *emparelhamento* M é um subconjunto de arestas duas a duas não adjacentes. As arestas pertencentes a M são ditas *M-emparelhadas*, ou simplesmente *emparelhadas*, enquanto que um vértice incidente a alguma aresta *M-emparelhada* é dito *M-saturado*, e aqueles não incidentes às arestas de M são *M-expostos*. Os grafos da Figura 14a e da Figura 14b ilustram emparelhamentos cujas arestas estão representadas em negrito. A aresta $(1,3)$ não está *M-emparelhada* na Figura 14a, porém o está na Figura 14b. O vértice 6 está *M-exposto* na Figura 14a, porém está *M-saturado* na Figura 14b e na Figura 14c. Neste último, todos os vértices estão *M-saturados*.

Observe que um conjunto vazio define um emparelhamento. Por outro lado, se $M \subseteq E$ é um emparelhamento qualquer $M' \subseteq M$ também é.

O emparelhamento M é maximal se qualquer aresta não emparelhada por M possuir uma das extremidades em M . Isto é, não existe emparelhamento M' , tal que $M' \supset M$. Além disso, se M for o emparelhamento de maior cardinalidade do grafo, então M é *máximo*, isto é, não existe emparelhamento M' , tal que $|M'| > |M|$. Finalmente, dizemos que M é *perfeito* quando todo vértice do grafo for saturado por M . Naturalmente, todo emparelhamento perfeito é máximo, e todo emparelhamento máximo é maximal. Um problema clássico de emparelhamentos consiste em determinar um emparelhamento máximo em um grafo e sua respectiva cardinalidade. A Figura 14a apresenta um emparelhamento maximal, porém não máximo do grafo. Este último é mostrado na Figura 14b. O emparelhamento do grafo da Figura 14c é perfeito, enquanto que o grafo da Figura 14a não admite emparelhamento perfeito.

Além disso, é importante apresentar o conceito de caminhos alternantes, o qual é central no estudo de algoritmos para a determinação de emparelhamentos máximos.

Seja G um grafo e M um emparelhamento de G . Um caminho *M-alternante* em G é um caminho cujas arestas estão alternadamente em $E - M$ e em M . Um caminho *M-alternante* cujos vértices extremos são ambos *M-expostos* é dito *M-aumentante*. Observe que um caminho *M-aumentante* possui uma quantidade par de vértices. A Figura 15a ilustra os conceitos apresentados. O emparelhamento considerado é $M = \{(2,3), (4,5), (7,8)\}$. O caminho $1, 2, 3, 4, 5$ é *M-alternante*, porém não *M-aumentante*, enquanto $1, 2, 3, 4, 5, 6$ é um caminho *M-aumentante*. Através da operação de diferença simétrica entre as arestas de M e as do caminho *M-aumentante* apresentadas, transforma-se o emparelhamento M no emparelhamento $M' = \{(1,2), (3,4), (5,6), (7,8)\}$, de cardinalidade uma unidade maior. Os emparelhamentos M e M' são ilustrados na Figura 15a e na Figura 15b, respectivamente.

Observe que determinar emparelhamentos que sejam maximais é uma questão bastante simples; pode ser resolvida através de algoritmo guloso, em tempo linear no tamanho do

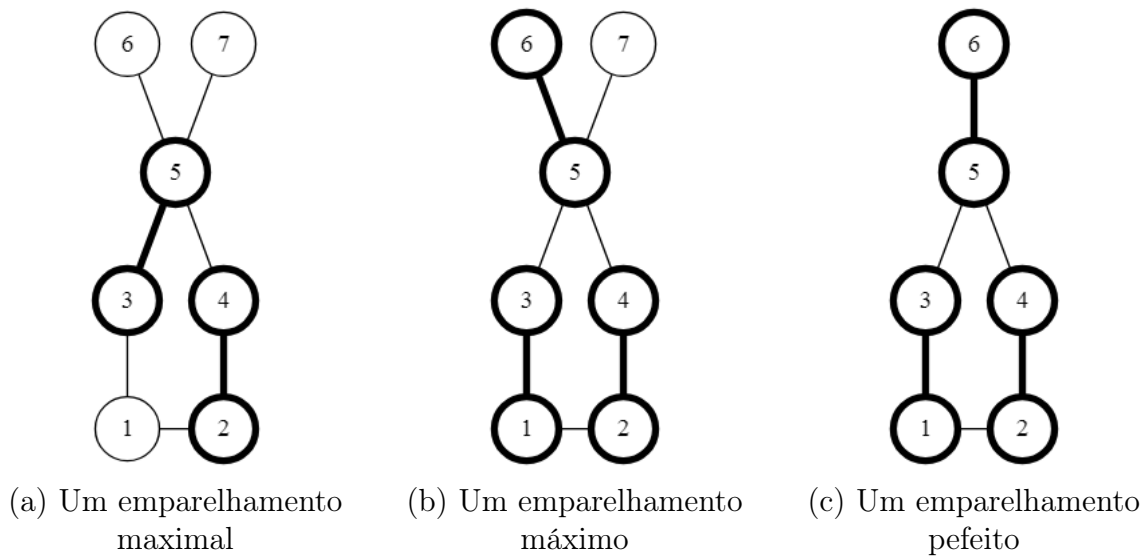


Figura 14 – Três emparelhamentos em grafos

grafo. A determinação de emparelhamentos máximos, contudo, é mais envolvente. Para esta última tarefa, aplica-se o Teorema 6, de Berge, que caracteriza emparelhamentos de cardinalidade máxima através de caminhos aumentantes.

Teorema 6. Seja $G(V,E)$ um grafo qualquer e M um emparelhamento de G . Então M possui cardinalidade máxima se e somente se G não possui caminho M -aumentante (BERGE, 1957).

Demonstração. Seja M um emparelhamento em G . Suponha que G possui um caminho M -aumentante P . Observe que P tem, por definição, um número par de vértices, digamos $P = v_0, v_1, \dots, v_{2k+1}$. Defina $M' \subseteq E$ através de $M' = M \setminus \{(v_1, v_2), (v_3, v_4), \dots, (v_{2k-1}, v_{2k})\} \cup \{(v_0, v_1), (v_2, v_3), \dots, (v_{2k}, v_{2k+1})\}$. Temos que M' é um emparelhamento em G com $|M'| = |M| + 1$ arestas e, portanto, M não tem cardinalidade máxima em G . Logo, se M possui cardinalidade máxima em G , então G não possui caminho M -aumentante. Por outro lado, suponha que M não tem cardinalidade máxima em G . Seja M' um emparelhamento de cardinalidade máxima. Logo $|M'| > |M|$. Denote por $M \Delta M'$ a diferença simétrica de M e M' , i.e., o conjunto das arestas que estão em $M \cup M'$ mas não estão em $M \cap M'$. Seja $H = G[M \Delta M']$. Cada vértice em H tem grau 1 ou 2, já que pode ser incidente a no máximo uma aresta de M e a uma aresta de M' . Logo cada componente conexa de H é um ciclo par com arestas alternadamente em M e M' ou um caminho com arestas alternadamente em M e M' . Mas H tem mais arestas de M' do que de M e, portanto, alguma componente que é um caminho Q em H deve começar e terminar com arestas de M' . Como a origem e o término de Q são M -saturados em H , segue que são vértices M -expostos em G . Logo Q é o caminho M -aumentante procurado. Portanto, se M não tem cardinalidade máxima em G , então G possui caminho aumentante em relação a M . \square

O problema de emparelhamentos tem sido estudado em larga escala pela comunidade científica e temos alguns resultados como algoritmos de complexidade de tempo $O(m\sqrt{n})$ para grafos bipartidos não ponderados (HOPCROFT; KARP, 1973). Esta complexidade de tempo foi posteriormente estendida para grafos gerais (MICALI; VAZIRANI, 1980).

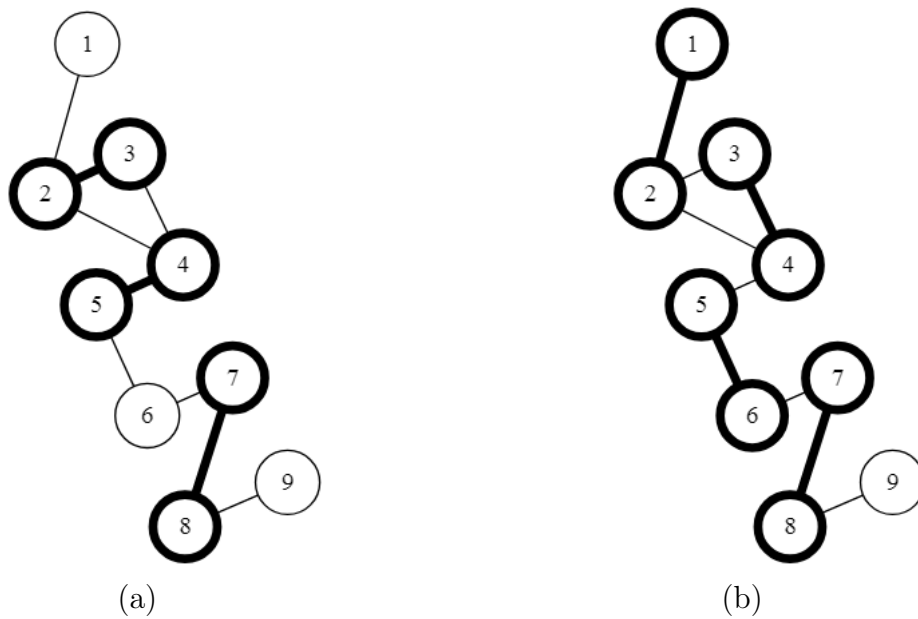


Figura 15 – Caminhos aumentante(a) e alternante(b)

2.1 Emparelhamentos restritos a subgrafos

Uma variação importante do problema do emparelhamento consiste em considerar emparelhamentos especiais conhecidos como *emparelhamentos restritos a subgrafos*, descritos em (GODDARD et al., 2005).

Dado um emparelhamento M , usaremos a notação $G[M]$ para representar o subgrafo induzido pelos vértices incidentes das arestas de M . Definimos M como sendo um \mathcal{P} -emparelhamento se $G[M]$ tem a propriedade \mathcal{P} , sendo \mathcal{P} uma propriedade de grafos. Denotamos $\beta_{\mathcal{P}}(G)$ pela máxima cardinalidade de um \mathcal{P} -emparelhamento. Além disso, definimos um \mathcal{P} -emparelhamento maximal como aquele que não está contido em um \mathcal{P} -emparelhamento maior.

Vamos mostrar alguns tipos de emparelhamentos restritos a subgrafos e exemplos baseados no grafo da Figura 16.

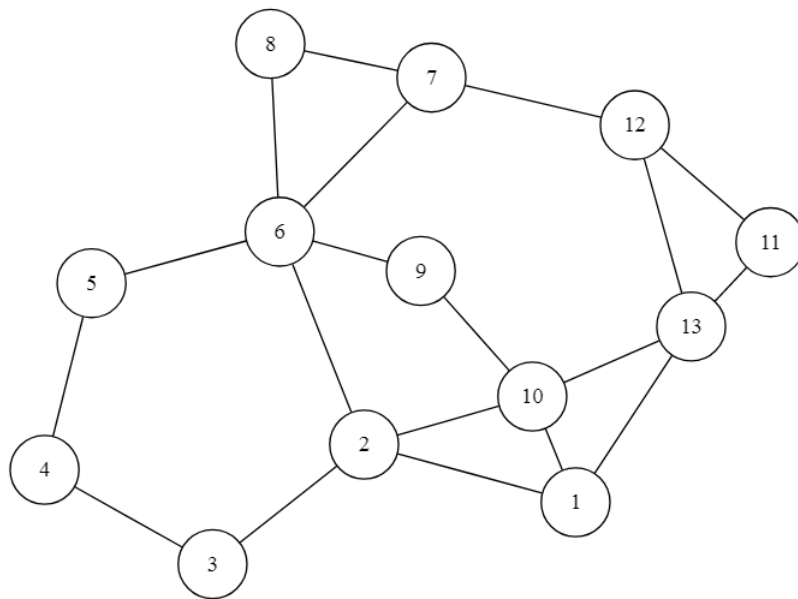


Figura 16 – Um grafo de exemplo

- Se a propriedade \mathcal{P} é a de ser um grafo qualquer, então um \mathcal{P} -emparelhamento é um emparelhamento irrestrito. Denotaremos um emparelhamento máximo desse tipo como $M(G)$ e sua cardinalidade como $\beta(G)$. Sempre que nos referirmos a um emparelhamento neste trabalho sem especificar sua propriedade, estaremos nos referindo a um emparelhamento irrestrito.
- Um emparelhamento M é chamado de *forte* ou *induzido* se não existem duas arestas de M incidentes a um mesmo vértice. Isto é, $G[M]$ é 1-regular. A cardinalidade de um emparelhamento induzido máximo é denotada por $\beta_*(G)$. Observe um emparelhamento forte ou induzido do grafo exemplo na Figura 17.

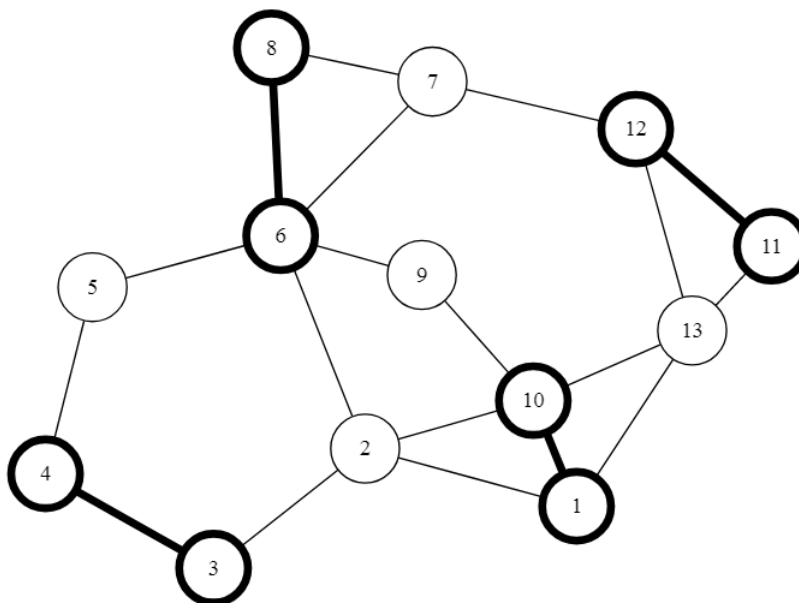


Figura 17 – Um emparelhamento desconexo, induzido, unicamente restrito e acíclico no grafo da Figura 16

O problema relativo à determinação de um emparelhamento máximo induzido pertence à classe NP-Completo para grafos bipartidos (CAMERON, 1989). Apesar disso, alguns autores mostraram que o problema pode ser resolvido em tempo polinomial para grafos trapezoidais (GOLUMBIC; LEWENSTEIN, 2000), grafos de comparabilidade (GOLUMBIC; LEWENSTEIN, 2000), grafos arco-circulares (GOLUMBIC; LASKAR, 1993) e tempo linear para grafos de intervalo (GOLUMBIC; LEWENSTEIN, 2000) e grafos cordais (BRANDSTÄDT; HOÀNG, 2008). Também foi mostrado que, para uma árvore T , é possível calcular $\beta_*(T)$ em tempo linear (FRICKE; LASKAR, 1992).

- Um emparelhamento M é chamado de *unicamente restrito* se $G[M]$ tem exatamente um emparelhamento máximo M . A cardinalidade de um emparelhamento máximo unicamente restrito é denotada por $\beta_{ur}(G)$. Observe um emparelhamento unicamente restrito do grafo exemplo na Figura 18.

Esse problema foi resolvido em tempo polinomial para grafos de intervalo (FRANCIS; JACOB; JANA, 2018).

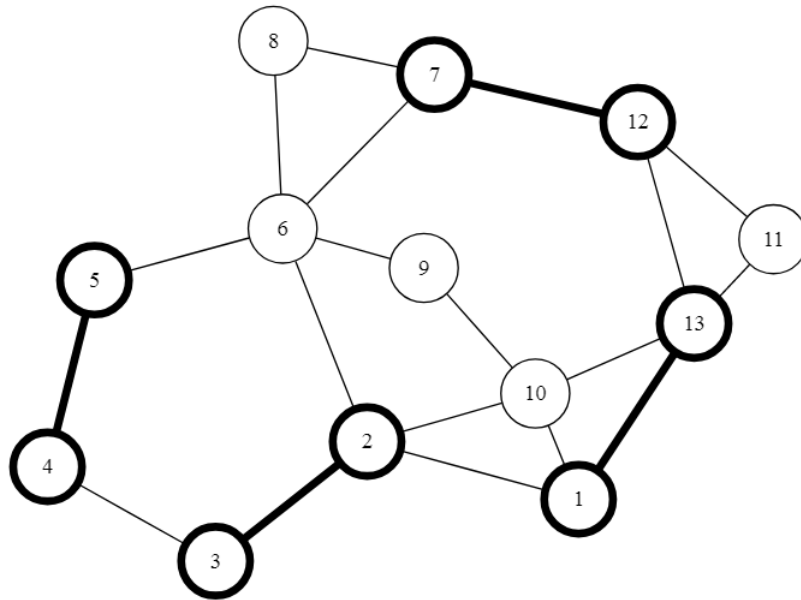


Figura 18 – Um emparelhamento acíclico, conexo, unicamente restrito e livre de isolamentos no grafo da Figura 16

- Um emparelhamento M é chamado de *conexo* se $G[M]$ é conexo. A cardinalidade de um emparelhamento máximo conexo é denotada como $\beta_c(G)$. Esse emparelhamento e sua cardinalidade são os correspondentes aos de um componente de G . Observe um emparelhamento conexo do grafo exemplo na Figura 18.

O problema de determinar um emparelhamento conexo foi resolvido em tempo polinomial para grafos cordais e foi provado ser NP-completo para grafos bipartidos (CAMERON, 2003). Um outro resultado importante foi a prova da igualdade $\beta_c(G) = \beta_{if}(G) = \beta(G)$, para um grafo conexo (GODDARD et al., 2005).

- Um emparelhamento M é chamado de *livre de isolamentos* se $|M| = 1$ ou $G[M]$ não possui um componente K_2 . A cardinalidade do emparelhamento máximo livre de isolamentos é denotado por $\beta_{if}(G)$. Esse parâmetro também é dado diretamente

pelas cardinalidades dos emparelhamentos máximos dos componentes de G . Observe um emparelhamento livre de isolamentos do grafo exemplo na Figura 18.

- Um emparelhamento M é chamado de *acíclico* se $G[M]$ é acíclico. A cardinalidade de um emparelhamento máximo acíclico é denotada por $\beta_{ac}(G)$. Observe um emparelhamento acíclico do grafo exemplo na Figura 18.

O problema do emparelhamento acíclico, pertencente à classe de problemas NP-difícil (GODDARD et al., 2005). Ele foi resolvido em tempo polinomial para grafos livres de P_4 e livres de $2P_3$ (FUERST; RAUTENBACH, 2017). Também há algoritmos lineares para grafos de cadeia e grafos bipartido de permutação (PANDA; PRADHAN, 2012).

- Um emparelhamento M é chamado de *desconexo* se $|M| = 1$ ou $G[M]$ é desconexo. Denotaremos um emparelhamento máximo desconexo por $M_d(G)$ e sua cardinalidade por $\beta_d(G)$. Observe emparelhamentos desconexos na Figura 19 e na Figura 17.

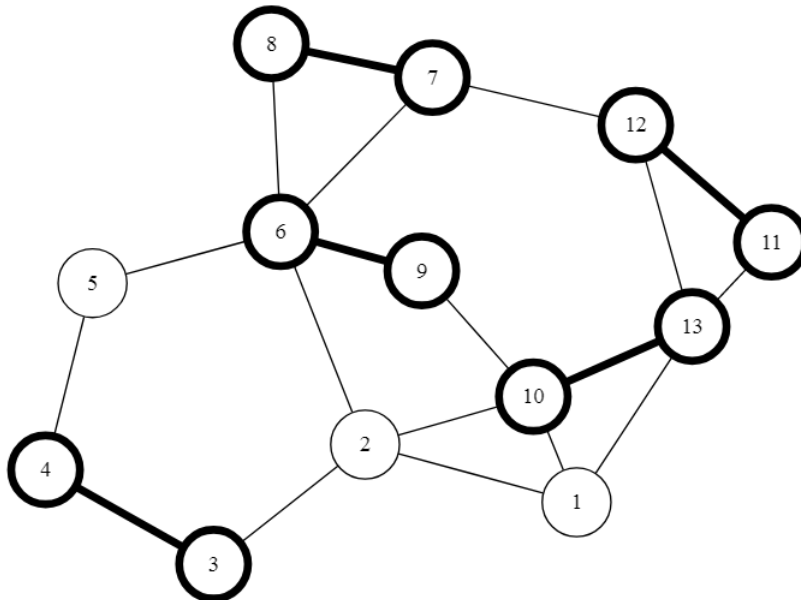


Figura 19 – Um emparelhamento desconexo no grafo da Figura 16

Neste trabalho, focaremos o estudo em emparelhamentos desconexos, que serão mais detalhados no próximo capítulo.

3 EMPARELHAMENTOS DESCONEXOS

Neste capítulo, serão apresentados estudos já existentes sobre emparelhamentos desconexos, assim como algoritmos novos, juntamente com suas bases teóricas, propostos para algumas classes de grafos.

Inicialmente, definiremos os limites inferiores e superiores da cardinalidade máxima de um emparelhamento desconexo em um grafo qualquer. Sabemos que, para duas propriedades de grafos \mathcal{L} e \mathcal{P} , se $\mathcal{P} \subseteq \mathcal{L}$, então $\beta_{\mathcal{P}} \leq \beta_{\mathcal{L}}$ (GODDARD et al., 2005). Mais especificamente, temos que, para um grafo qualquer $G(V, E)$, a propriedade de $G[M]$ ser induzido é uma restrição à propriedade de $G[M]$ ser desconexo, que por sua vez é uma restrição à propriedade de $G[M]$ ser simplesmente um grafo qualquer. Portanto, temos que $\beta_* \leq \beta_d \leq \beta$.

Observe que determinar um emparelhamento desconexo máximo de um grafo G pode ser simples quando o grafo possui uma das propriedades a seguir.

- Se G não tem arestas, então o emparelhamento desconexo não está definido.
- Se G é desconexo e contém mais de uma componente conexa não trivial, então $\beta(G) = \beta_d(G)$ e $M(G)$ também é $M_d(G)$.

Quando um grafo contém exatamente uma componente conexa não trivial, a determinação de emparelhamentos desconexos máximos pode não ser fácil. É possível deduzir que, se um grafo G possui uma única componente conexa não trivial C , $\beta_d(G) = \beta_d(C)$ e $M_d(G)$ é $M_d(C)$. Porém, ainda persiste um problema de emparelhamento desconexo máximo, o qual aparentemente não possui solução imediata. Por isso, o enfoque deste trabalho no estudo de emparelhamentos desconexos a seguir é em grafos conexos não triviais.

Definição 1. Em um grafo $G(V, E)$, definimos *separador não trivial* como um separador que deixa pelo menos duas componentes conexas não triviais ou um único K_2 , ou seja, apenas dois vértices adjacentes.

Usaremos $R_n(G)$ para denotar a menor cardinalidade de um separador não trivial. Temos que $\beta_d(G) \geq \beta(G) - R_n(G)$ (GODDARD et al., 2005). Mostraremos, a seguir, um método para calcular $\beta_d(G)$ em um grafo geral.

Os separadores não triviais são a chave para um algoritmo que obtém os emparelhamentos desconexos. Pois, se M é um emparelhamento desconexo, os vértices não saturados formam um separador não trivial, que contém um separador não trivial minimal. Assim, se os separadores não triviais minimais do grafo puderem ser enumerados em tempo polinomial, então β_d pode ser calculado em tempo polinomial. Basta considerar cada um dos separadores não triviais K e, por sua vez, calcular $\beta(G - K)$, o que pode ser feito em tempo polinomial. O maior deles é exatamente $\beta_d(G)$.

Por exemplo, seja a classe de grafos de intervalo. Cada grafo da classe tem $O(n)$ separadores não triviais minimais, que podem ser gerados com eficiência. Os grafos cordais têm $O(n)$ separadores minimais, e também podem ser determinados com eficiência (BRANDSTÄDT; LE; SPINRAD, 1999) (CHANDRAN, 2001), pois a ideia pode ser adaptada para separadores não triviais. Da mesma forma, os grafos arco circulares têm $O(n^2)$ separadores não triviais minimais, que também podem ser enumerados eficientemente (SUNDARAM; SINGH; RANGAN, 1994).

Consideraremos a definição de componentes não triviais como componentes conexas com pelo menos dois vértices. Usaremos também a definição abaixo nos estudos deste capítulo.

Definição 2. Definimos *articulação composta* em um grafo $G(V, E)$ como uma articulação $v \in V(G)$ cuja remoção deixa pelo menos duas componentes não triviais.

A seguir apresentamos alguns resultados de emparelhamentos desconexos para algumas classes de grafos.

3.1 Emparelhamento desconexo em caminhos

Nesta seção, definiremos $\beta_d(P_n)$ e $M_d(P_n)$ para um caminho $P_n(v_1, \dots, v_n)$.

Especificamente para um caminho, temos que todos os separadores minimais que deixam pelo menos duas componentes conexas não triviais são articulações compostas. Nesse caso, toda articulação composta v_i , para que deixe duas componentes não triviais ao seu corte é tal que $3 \leq i \leq n - 2$. Portanto, não existem articulações compostas para caminhos com $2 \leq n \leq 4$, o que faz com que $\beta_d(P_n) = 1$ nesse caso e $M_d(P_n)$ possa ser uma aresta qualquer.

O teorema a seguir generaliza os casos restantes, em que $n \geq 5$.

Teorema 7. Em um caminho P_n , $n \geq 5$ temos que $\beta_d(P_n) = \lfloor (n - 1)/2 \rfloor$.

Demonstração. Precisamos considerar caminhos de dois tipos: com comprimento par e com comprimento ímpar.

Inicialmente, consideremos o caso em que n é ímpar. Em um caminho P_n , de comprimento par (n ímpar), ao removermos uma articulação composta v_i , temos duas possibilidades. A primeira possibilidade é quando i é par, o que faz com que a remoção de v_i deixe dois caminhos de comprimento par, o de v_1 até v_{i-1} e o de v_{i+1} até v_n . Nesse caso, temos que em $M(P_n - v_i)$ há dois vértices não saturados. A segunda possibilidade é quando i é ímpar, o que faz com que a remoção de v_i deixe dois caminhos de comprimento ímpar. Nesse caso, temos que $M(P_n - v_i)$, i ímpar, é um emparelhamento perfeito e, portanto, é máximo e estritamente maior que $M(P_n - v_i)$, i par. Portanto, para um caminho de comprimento par, $n \geq 5$, $M_d(P_n)$ pode ser definido como $M(P_n - v_i)$, $1 < i < n$, ímpar, sendo v_i uma articulação composta. Portanto, $\beta_d(P_n) = (n - 1)/2 = \lfloor (n - 1)/2 \rfloor$.

Em um caminho de comprimento ímpar, n par, ao removermos qualquer articulação composta v_i , uma das duas componentes restantes é um caminho de comprimento ímpar e a outra é um caminho de comprimento par. Então, $\beta(P_n - v_i) = \beta(P_n) - 1$. Como $M(P_n)$ é um emparelhamento perfeito, então $\beta(P_n) = n/2$ e, ao removermos qualquer vértice de P_n , temos que a cardinalidade do emparelhamento diminui em 1. Portanto, para um caminho de comprimento ímpar, $n \geq 5$, $M_d(P_n)$ pode ser definido como $M(P_n - v_i)$, $2 < i < n - 1$, sendo v_i uma articulação composta. Portanto, $\beta_d(P_n) = n/2 - 1 = \lfloor (n - 1)/2 \rfloor$ \square

A Figura 20 ilustra um caminho de comprimento 6. Note que as articulações compostas desse exemplo são 3, 4 e 5. É possível, então, remover os vértices 3 ou 5 para obter um emparelhamento desconexo máximo, que tem tamanho 3. Por exemplo, ao removermos o vértice 3, obteremos o emparelhamento desconexo máximo formado pelas arestas (1, 2), (4, 5) e (6, 7). A remoção de 4 traz um emparelhamento maximal de tamanho 2, que não é máximo.



Figura 20 – O caminho de comprimento 6

Já a Figura 21 ilustra um caminho de comprimento 5. Note que as articulações compostas desse exemplo são 3 e 4 e suas remoções levam a emparelhamentos desconexos máximos, que têm tamanho 2. Por exemplo, ao removermos o vértice 4, podemos obter o emparelhamento desconexo máximo formado pelas arestas (2, 3) e (5, 6).

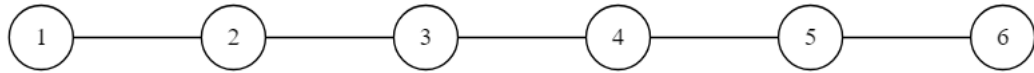


Figura 21 – O caminho de comprimento 5

3.2 Emparelhamento desconexo em ciclos

Nesta seção, definiremos $\beta_d(C_n)$ e $M_d(C_n)$ para um ciclo $C_n(v_1, \dots, v_n)$.

Especificamente em um ciclo, temos que todos os separadores minimais que deixam pelo menos duas componentes conexas não triviais são exatamente dois vértices com distância $3 \leq d \leq \lfloor n/2 \rfloor$ entre si. Portanto, é necessário termos $n \geq 6$. Ou seja, não existem separadores não triviais para ciclos com $3 \leq n \leq 5$, o que faz com que $\beta_d(C_n) = 1$ nesse caso e $M_d(C_n)$ possa ser uma aresta qualquer.

O teorema a seguir generaliza os casos restantes, em que $n \geq 6$.

Teorema 8. Em um ciclo C_n , $n \geq 6$, temos que $\beta_d(C_n) = \lfloor (n - 2)/2 \rfloor$.

Demonstração. Precisamos considerar ciclos de dois tipos: com comprimento par e com comprimento ímpar.

Inicialmente, consideremos o caso em que n é par, sendo par o comprimento. Ao retirarmos o primeiro vértice v_i , obteremos um caminho de comprimento ímpar, com $n - 1$ vértices. Resta, então, retirar uma articulação composta v_j desse caminho, tal que $3 \leq d(v_i, v_j) \leq \lfloor n/2 \rfloor$ no ciclo. Então, pelo Teorema 7, temos que, para um ciclo par C_n , $n \geq 6$, $\beta_d(C_n) = n/2 - 1 = \lfloor (n - 2)/2 \rfloor$.

Para ciclos de comprimento ímpar, no qual n é ímpar, ao retirarmos o primeiro vértice v_i , obteremos um caminho de comprimento par com $n - 1$ vértices. Resta, então, retirar uma articulação composta v_j desse caminho, tal que $3 \leq d(v_i, v_j) \leq \lfloor n/2 \rfloor$ no ciclo. Então, pelo Teorema 7, temos que, para um ciclo ímpar C_n , $n \geq 6$, $\beta_d(C_n) = \lfloor ((n - 1) - 1)/2 \rfloor = \lfloor (n - 2)/2 \rfloor$. \square

A seguir, a Figura 22(a) ilustra um ciclo de comprimento 6. Um possível par de vértices a se remover são 1 e 4, o que faz com que o par de arestas (2, 3) e (5, 6) forme um emparelhamento desconexo máximo, que tem tamanho 2.

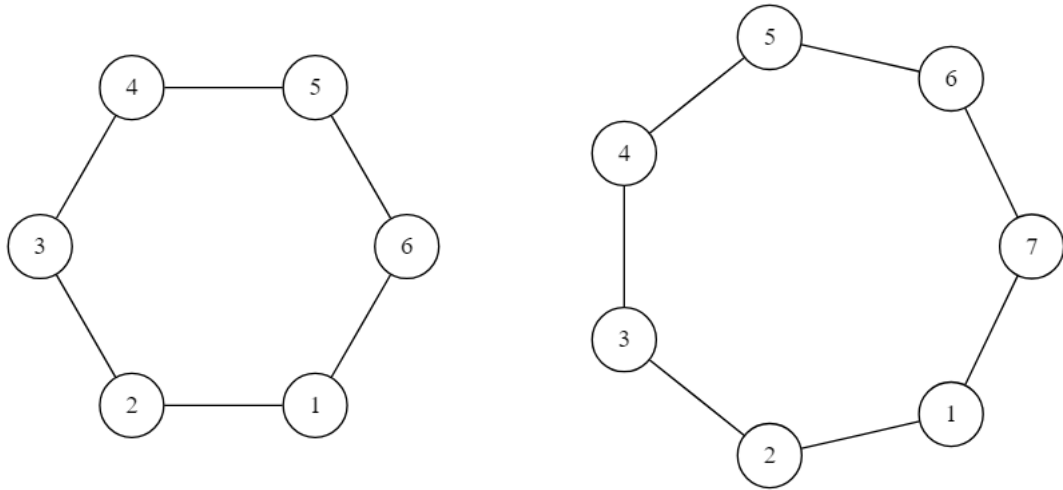


Figura 22 – Os ciclos $C_6(a)$ e $C_7(b)$

Já a Figura 22(b) ilustra um ciclo de comprimento 7. Um possível par de vértices a se remover são 2 e 5, o que faz com que o par de arestas $(3, 4)$ e $(6, 7)$ forme um emparelhamento desconexo máximo, que tem tamanho 2.

3.3 Emparelhamento desconexo em grafos split

Nesta seção, definiremos $\beta_d(G)$ e $M_d(G)$ para um grafo split G e, consequentemente, para um grafo completo K_n .

Teorema 9. Em um grafo split G , $n > 1$, temos que $\beta_d(G) = 1$.

Demonstração. Um grafo split G com n vértices, $n > 1$, é $(n - 1)$ -conexo, ou seja, não há separador que desconecte G em duas componentes conexas. Desse modo, o emparelhamento desconexo é sempre uma aresta qualquer. \square

Um caso específico é o de grafos completos. Como todo grafo completo é um grafo split, então para K_n , $n > 1$, vale que $\beta_d(K_n) = 1$ e $M_d(G)$ é uma aresta qualquer.

3.4 Emparelhamento desconexo em árvores

Nesta seção, apresentaremos resultados para a determinação de $\beta_d(T)$ e $M_d(T)$ para uma árvore qualquer T . Particularmente, em uma árvore, todos os separadores minimais são articulações, isto é, um único vértice que não é folha. Temos também que uma articulação composta em uma árvore é um vértice conectado a pelo menos dois outros vértices internos.

Teorema 10. $\beta(T) - 1 \leq \beta_d(T) \leq \beta(T)$ (GODDARD et al., 2005).

Demonstração. Já mostramos que $\beta_d(T) \leq \beta(T)$ para qualquer árvore T . Temos que $\beta(T) \leq \beta(T - v) + 1$ para qualquer vértice v . Se existe uma articulação composta v , o teorema é válido, pois $\beta_d(T - v) = \beta(T - v)$. Se não há articulação composta, então T é uma estrela ou uma estrela dupla e o teorema continua válido. \square

Com isso, resta saber como calcular $M_d(T)$ e como caracterizar quando $\beta_d(T) = \beta(T) - 1$ e quando $\beta_d(T) = \beta(T)$. Isso será desenvolvido a seguir.

Definição 3. Definimos uma *articulação composta livre* v em um grafo G se v é articulação composta e $\beta(G) = \beta(G - v)$.

Teorema 11. $\beta_d(T) = \beta(T)$ se e somente se $\beta(T) = 1$ ou T possui pelo menos uma articulação composta livre.

Demonstração. Caso T possua alguma articulação composta livre v , então $\beta_d(T) = \beta(T) = \beta(T - v)$. Se T possui alguma articulação composta u e não possui nenhuma articulação composta livre, então $\beta_d(T) = \beta(T) - 1 = \beta(T - u)$.

Resta identificar quando T não possui articulação composta. Neste caso, pelo Teorema 10, T é uma estrela ou uma estrela dupla. Se T é uma estrela, $\beta_d(T) = \beta(T) = 1$. Se T é uma estrela dupla, $\beta(T) = 2$ e $\beta_d(T) = \beta(T) - 1 = 1$.

Isso encerra todos os casos, o que faz o teorema válido. \square

Resumindo, temos os seguintes casos para $M_d(T)$ e $\beta_d(T)$:

- T possui alguma articulação composta v . Então
 - $\beta_d(T) = \beta(T)$, se T possuir alguma articulação composta livre u . Nesse caso, $M_d(T)$ pode ser determinado como $M(T - u)$.
 - $\beta_d(T) = \beta(T) - 1$, caso contrário. Nessa situação $M_d(T)$ pode ser descrito como $M(T - v)$.
- T não possui articulação composta. Então
 - $\beta_d(T) = \beta(T) = 1$, caso T seja uma estrela. Nessa situação $M_d(T)$ é uma aresta qualquer.
 - $\beta_d(T) = \beta(T) - 1 = 1$, caso contrário. T é uma estrela dupla e $M_d(T)$ é uma aresta qualquer.

A determinação de $M(T)$ pode ser feita em complexidade de tempo $O(n)$, (SAVAGE, 1980). Portanto, a dificuldade restante para se calcular $\beta_d(T)$ é identificar a existência ou não de articulações compostas livres. Isso será trabalhado a seguir.

Definição 4. Definimos v uma *articulação candidata* em uma árvore T^r como um vértice interno que satisfaça um dos tipos a seguir

1. Todos os filhos de v possuem pelo menos uma folha como filho.
2. Todos os filhos de v possuem pelo menos uma folha ou uma articulação candidata como filho.

Lema 4. Se v é uma articulação candidata na árvore T^r então $\beta(T_v^r) = \beta(T_v^r - v)$.

Para essa prova, vamos considerar que v , uma articulação candidata em T_v^r , tem n filhos (u_1, \dots, u_n) . Vamos chamar de w_i um dos filhos folha ou articulação candidata de cada u_i . A prova será por indução no número de descendentes de v em T_v^r .

Demonstração. Caso base: Seja v uma articulação candidata do tipo 1, ou seja, todo w_i é folha. Vamos construir um emparelhamento máximo M , para T_v^r como sendo a união de $\bigcup_{i=1}^n M(T_{u_i}^r - u_i)$ com $\bigcup_{i=1}^n \{(u_i, w_i)\}$. Vamos provar que esse é um emparelhamento máximo em T .

Como $M(T_{u_i}^r - u_i)$ é um emparelhamento máximo, então não existe caminho M -aumentante entre vértices desse conjunto. Também não há caminho M -aumentante entre vértices desse conjunto para vértices de fora desse conjunto, pois o caminho necessariamente teria que passar por u_i e terminar em w_i . Assim, o caminho seria no máximo M -alternante. Também não há caminho M -aumentante partindo de v pelo mesmo motivo. Portanto, não há caminho M -aumentante em T e M é máximo e v não está saturado.

Passo indutivo: Supondo que para qualquer articulação candidata, vale o teorema, vamos provar que v sendo do tipo 2, então $\beta(T_v^r) = \beta(T_v^r - v)$.

Observamos que para todo w_i , vale que $\beta(T_{w_i}^r - w_i) = \beta(T_{w_i}^r)$. Ou seja, existe um emparelhamento máximo em $T_{w_i}^r$ em que w_i não estará saturado. Isso ocorre pois se w_i é folha, $T_{w_i}^r$ é o próprio w_i e portanto $\beta(T_{w_i}^r) = 0$. Se w_i é articulação candidata, então pela suposição do passo indutivo $\beta(T_{w_i}^r - w_i) = \beta(T_{w_i}^r)$.

Então, vamos construir um emparelhamento máximo M como sendo a união de $\bigcup_{i=1}^n M(T_{w_i}^r - w_i)$ com $\bigcup_{i=1}^n M(T_{u_i}^r - u_i - T_{w_i}^r)$ e com $\bigcup_{i=1}^n \{(u_i, w_i)\}$. Pelo mesmo motivo do caso base, não há caminho M -aumentante em T e portanto M é máximo e v não está saturado. \square

Teorema 12. Um vértice r é articulação composta livre em T se e somente se r é articulação composta e articulação candidata em T^r .

Demonstração. (\Rightarrow) :

Se r é articulação composta livre, então r é articulação composta. Resta mostrar que r é uma articulação candidata.

Considere um emparelhamento máximo M em T^r com r não emparelhado. Considere s_1, s_2, \dots, s_n os filhos de r . Para cada s_j , vamos construir um caminho $p_1, p_2, p_3, \dots, p_m$, onde $p_1 = s_j$, de p_1 até uma folha.

Temos que p_1 necessariamente está emparelhado com um vértice, que vamos chamar de p_2 . Caso p_2 não seja uma folha, p_3 será um vizinho qualquer de p_2 . Nesse caso, p_3 está saturado, pois ao contrário haveria um caminho M -aumentante de r a p_3 . Seguimos adiante com o método até chegar a uma folha. Generalizando, p_i com i ímpar pode ser qualquer vértice filho de p_{i-1} e sempre estará saturado pelo mesmo motivo de p_2 . Da mesma forma, p_i com i par ou é uma folha ou é uma articulação candidata e isso confirma a primeira parte da prova.

(\Leftarrow) :

Segundo o Lema 4, $\beta(T^r) = \beta(T^r - r)$. Como r é articulação composta e raiz de T , então $\beta(T) = \beta(T - r) = \beta_d(T)$, ou seja, r é uma articulação composta livre em T . \square

Lema 5. Sejam v, u, w, r vértices em T e as seguintes proposições:

- (i) Em T^r , v é articulação candidata.
- (ii) Em T^r , u é pai de v .
- (iii) Em T^v , w é folha ou articulação candidata.

(iv) Em T^v , w é filho de u .

(i) e (ii) e (iii) e (iv) $\Leftrightarrow v$ é articulação candidata em T^v .

Demonstração. (\Rightarrow) :

Observamos que, ao enraizarmos T em v ao invés de r , v terá somente mais um filho imediato em T^v que não possui em T^r , u .

Sabemos todo filho de v em T^r , vale que esse vértice possui uma articulação candidata ou folha como filho. Então, para que v seja articulação candidata em T^v , basta que o seu novo filho, u , possua também algum filho w folha ou articulação candidata em T^v , o que faz parte da condição.

(\Leftarrow) :

Se v é articulação candidata em T^v , então todos os filhos, incluindo u , possuem algum filho folha ou articulação candidata. \square

Apresentaremos a seguir um algoritmo recursivo que identifica todas as articulações compostas livres de uma árvore, com complexidade de tempo $O(n)$.

A ideia do algoritmo é usar duas buscas em profundidade em T . A primeira, a partir de um vértice qualquer v , encontra-se todas as articulações candidatas de T^v , das folhas à raiz, usando as propriedades do Teorema 3.4.

Ao final da primeira busca, pode se saber se v é articulação candidata em T^v , ou seja, uma articulação composta livre em T . A partir dessa informação e do Lema 5, a segunda busca identifica as demais articulações compostas livres em T , processando os vértices de T^v no sentido da raiz para as folhas.

A seguir, descreveremos mais detalhadamente as funções utilizadas.

- A função $AC(T)$ recebe uma árvore T e retorna o conjunto de articulações compostas de T . O algoritmo pode utilizar uma classificação de cada vértice de T como articulação composta segundo critério foi descrito anteriormente. Por esse motivo, o código não foi implementado.
- A função $CARD\text{EMP}IRRESTRITO\text{ARVORES}(T)$ recebe uma árvore T e retorna a cardinalidade de um emparelhamento máximo irrestrito em T , ou seja, retorna $\beta(T)$. Essa função pode ser obtida através do Teorema 3.4, cujo algoritmo pode ser feito como descrito em (SAVAGE, 1980).
- As funções $MARCA(v)$ e $DESMARCA(v)$ recebem um vértice v de T e, respectivamente, marcam e desmarcam v a partir de variáveis globais.
- A função $ACL(T)$ recebe uma árvore T e retorna o conjunto de articulações compostas livres em T .
- A função $BUSCA1(v)$ recebe um vértice de T e preenche parcialmente as listas de cada vértice de T , o que determina as articulações candidatas de T^v .
- A função $BUSCA2(v)$ recebe um vértice de T e preenche o restante das listas de cada vértice de T , o que determina as articulações compostas livres de T .

Algoritmo 3 – Cardinalidade do Emparelhamento Desconexo máximo em uma árvore

- 1: **função** $BUSCA1(\text{Vértice: } v)$
- 2: $MARCA(v)$

```

3:   para  $u$  vizinho não marcado de  $v$  faça
4:       se BUSCA1( $u$ ) então
5:           lista[ $v$ ]  $\leftarrow$  lista[ $v$ ]  $\cup \{u\}$ 
6:   retorna |lista[ $v$ ]| = 0
7:
8: função BUSCA2(Vértice:  $v$ )
9:   MARCA( $v$ )
10:  para  $u$  vizinho não marcado de  $v$  faça
11:      se |(lista[ $v$ ] -  $\{u\})| > 0$  então
12:          lista[ $u$ ]  $\leftarrow$  lista[ $u$ ]  $\cup \{v\}$ 
13:      BUSCA2( $u$ )
14:
15: função ACL(Árvore:  $T$ )
16:    $v \leftarrow$  Vértice qualquer de  $T$ 
17:    $r \leftarrow \{\}$ 
18:   para  $u$  vértice de  $T$  faça
19:       DESMARCA( $u$ )
20:       lista[ $v$ ]  $\leftarrow \{\}$ 
21:   BUSCA1( $v$ )
22:   para  $u$  vértice de  $T$  faça
23:       DESMARCA( $u$ )
24:   BUSCA2( $v$ )
25:   para  $v$  articulação composta de  $T$  faça
26:       se |lista[ $v$ ]| = 0 então
27:            $r \leftarrow r \cup \{v\}$ 
28:   retorna  $r$ 
29:
30: #Dados:  $T$ : Árvore
31:  $\beta(T) \leftarrow$  CARDEMPÍRRESTRITOARVORES( $T$ )
32: se |AC( $T$ )| = 0 então
33:     se  $\beta(T) \leq 1$  então
34:          $\beta_d(T) = \beta(T)$  e  $M_d(T)$  pode ser qualquer  $M(T)$ 
35:     senão
36:          $\beta_d(T) = \beta(T) - 1$  e  $M_d(T)$  é uma aresta qualquer
37: senão
38:     se |ACL( $T$ )| > 0 então
39:          $\beta_d(T) = \beta(T)$  e  $M_d(T)$  pode ser qualquer  $M(T - v)$ ,
40:         para qualquer articulação composta livre  $v$ .
41:     senão
42:          $\beta_d(T) = \beta(T) - 1$  e  $M_d(T)$  pode ser qualquer  $M(T - v)$ ,
43:         para qualquer articulação composta  $v$ .

```

A seguir, apresentaremos um exemplo de execução do algoritmo em que usaremos a árvore T descrita na Figura 23.

Como $\beta(T) > 1$, teremos que chamar a função ACL(T) para descobrir se existem articulações compostas livres e, se existirem, quais são elas. Em ACL(T), escolhemos o vértice v_1 como parâmetro para a chamada das funções *Busca1* e *Busca2*. Após a

Listas	Conteúdo	Listas	Conteúdo
$lista[v_{13}]$	$\{\}$	$lista[v_2]$	$\{v_4\}$
$lista[v_{11}]$	$\{v_{13}\}$	$lista[v_{10}]$	$\{\}$
$lista[v_{12}]$	$\{\}$	$lista[v_8]$	$\{v_{10}\}$
$lista[v_9]$	$\{v_{12}\}$	$lista[v_6]$	$\{\}$
$lista[v_7]$	$\{\}$	$lista[v_3]$	$\{v_6\}$
$lista[v_5]$	$\{v_7\}$	$lista[v_1]$	$\{\}$
$lista[v_4]$	$\{\}$		

Tabela 1 – Tabela com o conteúdo das listas após a execução de $BUSCA1(v_1)$

execução de cada uma das funções mencionadas, vamos mostrar como estão preenchidas as listas de cada vértice e as conclusões que podemos chegar.

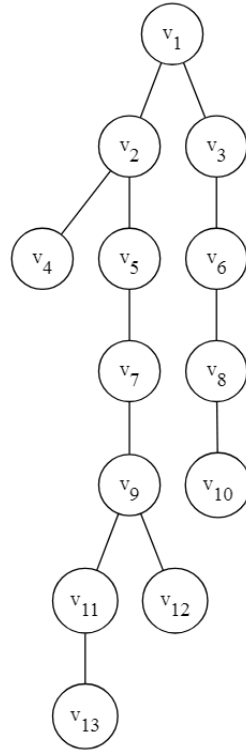


Figura 23 – Um exemplo de árvore T

Ao final de $BUSCA1(v_1)$, teremos as listas segundo a Tabela 1.

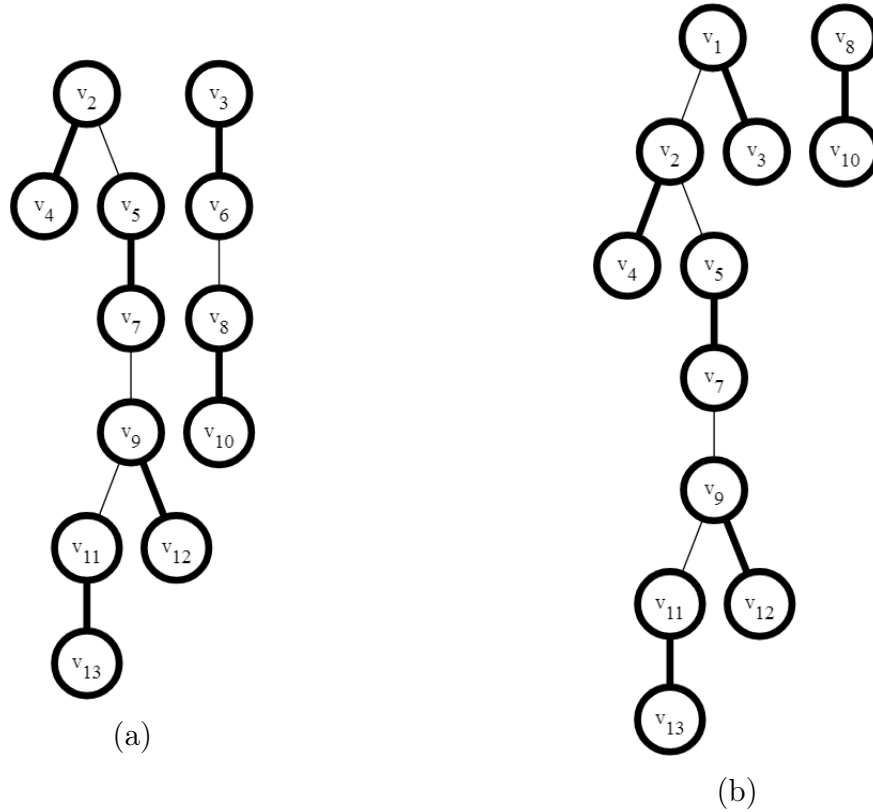
Note que os vértices cujas respectivas listas estão vazias são as articulações candidatas e folhas de T^{v_1} . Nesse caso, como a raiz v_1 é uma articulação composta e uma articulação candidata em T^{v_1} , já podemos concluir que v_1 também é uma articulação composta livre em T .

Ao final de $BUSCA2(v_1)$, teremos as listas segundo a Tabela 2.

Note que as listas de v_1 , v_4 , v_6 e v_{10} estão vazias. Desses quatro vértices, como somente v_1 e v_6 são articulações compostas, então v_1 e v_6 são as únicas articulações compostas livres em T .

Ao final do algoritmo, como existe pelo menos uma articulação composta livre, temos que $\beta(T) = \beta_d(T)$. Como v_1 e v_6 são as articulações compostas livres de T , $\beta_d(T) = \beta(T - v_1) = \beta(T - v_6)$. Portanto, $M_d(T)$ pode ser definido por $M(T - v_1)$ ou $M(T - v_6)$.

Listas	Conteúdo	Listas	Conteúdo
$lista[v_{13}]$	$\{v_{11}\}$	$lista[v_2]$	$\{v_4, v_1\}$
$lista[v_{11}]$	$\{v_{13}\}$	$lista[v_{10}]$	$\{\}$
$lista[v_{12}]$	$\{v_9\}$	$lista[v_8]$	$\{v_{10}, v_6\}$
$lista[v_9]$	$\{v_{12}\}$	$lista[v_6]$	$\{\}$
$lista[v_7]$	$\{v_5\}$	$lista[v_3]$	$\{v_6, v_1\}$
$lista[v_5]$	$\{v_7\}$	$lista[v_1]$	$\{\}$
$lista[v_4]$	$\{\}$		

Tabela 2 – Tabela com o conteúdo das listas após a execução de $BUSCA2(v_1)$ Figura 24 – Grafos induzidos por emparelhamentos desconexos máximos em T

Observe a seguir dois emparelhamentos desconexos máximos em T na Figura 24. No emparelhamento da Figura 24a, v_1 não está saturado e, no da Figura 24b, v_6 também não está.

Teorema 13. O Algoritmo 3 identifica todas as articulações compostas livres em T .

O algoritmo usa uma estrutura de árvore T^r , sendo r um vértice qualquer de T e é dividido em duas buscas em profundidade, sendo uma para cada sentido de percurso na árvore. A primeira analisa os caminhos das folhas à raiz e já é capaz de identificar se r é articulação composta livre. A segunda analisa os caminhos da raiz às folhas e identifica as demais articulações compostas. Cada vértice de v em T possui uma lista que será chamada de $lista[v]$ ao longo do algoritmo.

Durante a primeira busca, são identificadas em T^r as suas respectivas articulações candidatas com uma busca em profundidade a partir de r da seguinte forma: caso um vértice v seja articulação candidata ou folha, adiciona-se v à lista de seu pai. No caso

contrário, v necessariamente possui um filho folha ou articulação candidata filha e, assim, não é adicionado à lista de seu pai.

Ao final da primeira busca, r é articulação composta livre, se for articulação composta (Teorema 11) e articulação candidata em T^r . Resta analisar as outras articulações na próxima busca.

A segunda busca utiliza a ideia do Lema 5, pois já sabemos se r de fato é articulação candidata em T^r . Assim, para cada vizinho v_i de r em T , coloca-se r na lista de v_i se r possuir algum filho v_j , $i \neq j$, que possua alguma folha ou articulação candidata como filho. Essa informação já foi calculada na primeira busca e está guardada em $lista[r]$. Portanto, para determinar se v_i é articulação candidata em T^v , basta constatar se existe algum vértice em $lista[r] - v_i$. Aplica-se recursivamente esse procedimento em seus filhos até as folhas de T^r .

Ao final da segunda busca, todas as articulações compostas, cujas listas estão vazias são articulações compostas livres em T .

Teorema 14. O Algoritmo 3 pode ser implementado em complexidade de tempo $O(n)$.

Demonstração. O emparelhamento máximo irrestrito em uma árvore pode ser implementado em complexidade de tempo $O(n)$ (SAVAGE, 1980). A determinação das articulações compostas em $AC(T)$, conforme descrito anteriormente, pode ser obtida em complexidade de tempo $O(n)$. As funções $BUSCA1(v)$ e $BUSCA2(v)$ podem ser implementadas em complexidade de tempo $O(n)$ por conta da marcação dos vértices. Todas as manipulações de conjuntos podem ser realizadas a partir de listas de acesso direto com tamanho igual ao número de vértices. Em $ACL(T)$, todos os loops têm complexidade $O(n)$, o que leva à conclusão que a determinação de um emparelhamento desconexo máximo assim como sua cardinalidade tem complexidade $O(n)$ em uma árvore. \square

Conforme exposto anteriormente, temos que tanto $\beta_d(T)$ quanto $M_d(T)$ podem ser calculados em complexidade de tempo $O(n)$.

3.5 Emparelhamento desconexo em grafos bloco

Nesta seção, apresentaremos os estudos feitos para a determinação de $\beta_d(B)$ e $M_d(B)$ para um grafo bloco B .

Teorema 15. Em um grafo bloco B , todo separador minimal que deixa pelo menos duas componentes não triviais é uma articulação composta.

Demonstração. Para um grafo bloco B , todo separador minimal é uma articulação. Isso acontece pois esses grafos são construídos a partir de componentes biconexas que se interceptam em apenas um vértice. Essas interseções são exatamente as articulações no grafo. Portanto, para um separador cuja remoção produz pelo menos duas componentes não triviais, basta que a articulação seja composta. \square

Então, para reconhecer se $v \in B$ é uma articulação composta, basta analisar as componentes biconexas de que v faz parte. Se houver pelo menos duas que possuem mais de uma articulação ou possuem mais de 2 vértices, então v é uma articulação composta.

Lema 6. Um grafo bloco B não possui articulação composta se e somente se B é um grafo split em que todos os vértices do conjunto independente têm grau 1.

Demonstração. (\Rightarrow):

Se B não possui articulação composta, só pode existir no máximo um bloco de tamanho maior que 2. Caso contrário, há pelo menos uma articulação composta no caminho de dois desses blocos.

Sabemos que se todos os blocos em B possuem tamanho $t = 2$, então B é uma árvore. Nesse caso, se B não possui articulação composta, B é uma estrela ou estrela dupla, de acordo com o Capítulo 1. Em ambos os casos, B é um grafo split em que todas as folhas formam um conjunto independente e têm grau 1.

Se B possui um bloco C de tamanho $t > 2$, consideramos o conjunto S formado pelos blocos de tamanho 2 ligados a C . Nesse caso, $|S| \geq 0$. Observe que todo $s_i \in S$, é formado por dois vértices, um deles uma articulação e outro um vértice de grau 1. O primeiro é o vértice que pertence a C e o segundo não possui nenhum vizinho além do primeiro, pois caso possuísse, a remoção do primeiro em B deixaria duas componentes não triviais. Nesse caso, temos que B é um grafo split em que a clique é C , e o conjunto independente é formado pelos vértices de grau 1 de S .

(\Leftarrow):

Se B é um grafo split em que todos os vértices do conjunto independente têm grau 1, então vamos provar que B é um grafo bloco. Vamos considerar a clique de B como um bloco C do grafo bloco. Além disso, vamos considerar cada vértice do conjunto independente e seu vizinho, que pertence a C , como um bloco de B de tamanho 2. Temos então que B é um grafo bloco e não possui articulação composta pois as únicas articulações são os vizinhos dos vértices do conjunto independente. A remoção delas só pode deixar no máximo uma componente não trivial. □

Teorema 16. Seja B um grafo bloco sem articulação composta. Então, $\beta_d(B) = 1$

Demonstração. Temos que pelo Lema 6 e o Teorema 9, como B é um grafo split, então $\beta_d(B) = 1$ e $M_d(B)$ é uma aresta qualquer. □

Teorema 17. Se B é um grafo bloco com articulação composta, então $\beta(B) - 1 \leq \beta_d(B) \leq \beta(B)$.

Demonstração. De forma análoga à prova feita para árvores, precisamos mostrar que $\beta_d(B) \geq \beta(B) - 1$. Temos que $\beta(B) \leq 1 + \beta(B - v)$ para qualquer vértice v . Para o caso específico de uma articulação composta livre, $\beta_d(B - v) = \beta(B - v)$. □

Resumindo, temos os seguintes casos para $M_d(B)$ e $\beta_d(B)$:

- B possui alguma articulação composta v . Então
 - $\beta_d(B) = \beta(B)$, caso B possua alguma articulação composta livre u . Então, $M_d(B)$ pode ser calculado como $M(B - u)$.
 - $\beta_d(B) = \beta(B) - 1$, caso contrário. $M_d(B)$ pode ser calculado como $M(B - v)$.
- B não possui articulação composta. Então
 - $\beta_d(B) = 1$ e $M_d(B)$ é uma aresta qualquer.

Apresentaremos um algoritmo e sua base teórica para calcular $M_d(B)$ e $\beta_d(B)$. Para isso, usaremos uma árvore auxiliar T , obtida a partir de B , conforme descrito a seguir. Alguns vértices $v \in T$ possuem um peso que denotaremos como $\text{peso}(v)$. A criação de T baseia-se na identificação das componentes biconexas de B da seguinte maneira:

- $v \in V(T)$, se v é uma articulação em B .
- $C \in V(T)$ e $\text{peso}(C) = p$, se C é um bloco de B , sendo p o número de vértices simpliciais de C .
- $(v, C) \in E(T)$, se v foi originado de uma articulação em B e pertence à clique em B que originou o vértice C .

Assim, obtem-se uma árvore característica, que nesta seção chamaremos de $T(B)$, em que os vértices relativos às articulações de B são alternadas com vértices relativos a blocos de B . A árvore é tal que cada nível contém apenas vértices de um desses tipos. Usaremos também a notação B_b^a para representar o grafo induzido pelos vértices simpliciais e articulações em B que formaram a árvore $T(B)_b^a$.

A Figura 25 ilustra a construção de $T(B)^{v_{12}}$, a partir de um grafo bloco.

Observe que os níveis ímpares se referem às articulações de B enquanto os pares se referem aos blocos. Nesse caso, todas as folhas são vértices originados de um bloco de B . Também é possível concluir que não existe um vértice folha com peso 0, pois nesse caso esse bloco só possuiria um vértice que é a articulação pai.

As legendas acima dos vértices referem-se aos rótulos dos mesmos. Os vértices $v_{12}, v_{11}, v_{19}, v_{13}, v_{17}, v_7, v_{15}, v_8$ são originados, respectivamente, das articulações 12, 11, 19, 13, 17, 7, 15, 8 pertencentes ao grafo bloco. Da mesma forma, os vértices C_1, C_2, \dots, C_{13} são originados das cliques correspondentes no grafo bloco. As legendas dentro dos vértices referem-se aos pesos dos mesmos.

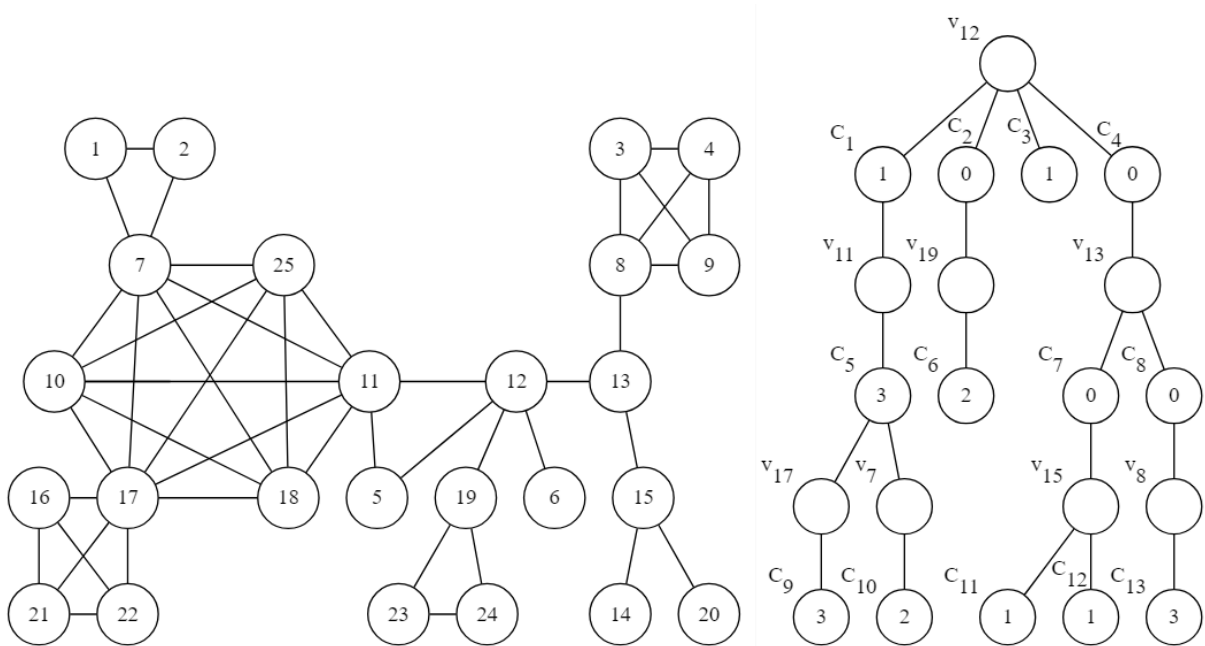


Figura 25 – Um grafo bloco e sua árvore auxiliar

Definição 5. Definimos v uma *articulação candidata* em uma árvore $T(B)^r$, onde r é um vértice originado de uma articulação de B , como um vértice que satisfaça um dos tipos a seguir

1. Todos os filhos de v são folhas e possuem peso par.
2. Para todo filho de v , o valor de seus respectivos pesos somado com número de articulações candidatas filhas é par.

Observamos que se uma articulação v não é uma articulação candidata, então v possui pelo menos um filho u tal que

1. u é folha e $\text{peso}(u)$ é ímpar
2. u não é folha e o valor de $\text{peso}(u)$ somado com o número de articulações candidatas filhas de u é ímpar.

Para uma melhor leitura, usaremos $\text{ac}(T, C)$ para denotar o conjunto de articulações candidatas filhas de C em uma árvore enraizada T .

Podemos observar que na Figura 25 há 4 articulações candidatas, v_7 , v_{11} , v_{13} e v_{19} pelos motivos a seguir:

- Os únicos filhos de v_7 e v_{19} são folhas, e têm peso par. Nesse caso, elas não possuem articulações candidatas filhas e 0 somado com um número par é par.
- Os dois filhos de v_{13} possuem peso 0 e 0 articulações candidatas filhas, o que faz com que a soma desses valores seja par. Além disso, o único filho de v_{11} possui peso 3 e 1 articulação candidata filha.

Observamos também que há 4 vértices que não são articulações candidatas, v_8 , v_{12} , v_{15} e v_{17} , pois possuem, cada um deles, um filho folha com peso ímpar.

Teorema 18. Seja v uma articulação de um grafo bloco B que possui articulação composta. Então, $\beta(B_v^r) = \beta(B_v^r - v)$ se e somente se v é uma articulação candidata em $T(B)_v^r$

Demonstração. Considere v uma articulação qualquer de B . Vamos provar que $\beta(B_v^r) = \beta(B_v^r - v)$ se v é uma articulação candidata e que $\beta(B_v^r) = \beta(B_v^r - v) + 1$ se v não é uma articulação candidata por indução na altura h de $T(B)_v^r$.

Considere $S = \{s_1, \dots, s_n\}$ os filhos de v em $T(B)_v^r$, que são as componentes correspondentes de B . Além disso, considere M um emparelhamento máximo em B_v^r .

Casos base: $h = 2$, isto é, para todo $s \in S$, s é folha.

- Se v é articulação candidata, então para todo $s \in S$, o $\text{peso}(s)$ é par e $|\text{ac}(T(B)_v^r, s)| = 0$. Vamos provar então que v pode estar exposto em M .

Já que o número de vértices de todas as componentes de S é ímpar, bastaria em M emparelhar todos os vértices entre si, exceto v . Como esse emparelhamento é perfeito em $B_v^r - v$, então ele é máximo em B_v^r . Isso prova que v pode ser M -exposto.

- Se v não é articulação candidata, então existe $s \in S$ tal que $\text{peso}(s)$ é ímpar e $|\text{ac}(T(B)_v^r, s)| = 0$. Vamos provar que $\beta(B_v^r) = \beta(B_v^r - v) + 1$.

Considere M_s o emparelhamento máximo em B_s^r . Como o número de vértices desse subgrafo é ímpar, um vértice ficará M_s -exposto. Como M é um emparelhamento máximo, v precisaria estar M -saturado. Caso contrário, existiria um caminho M -aumentante de v até o vértice M_s -exposto.

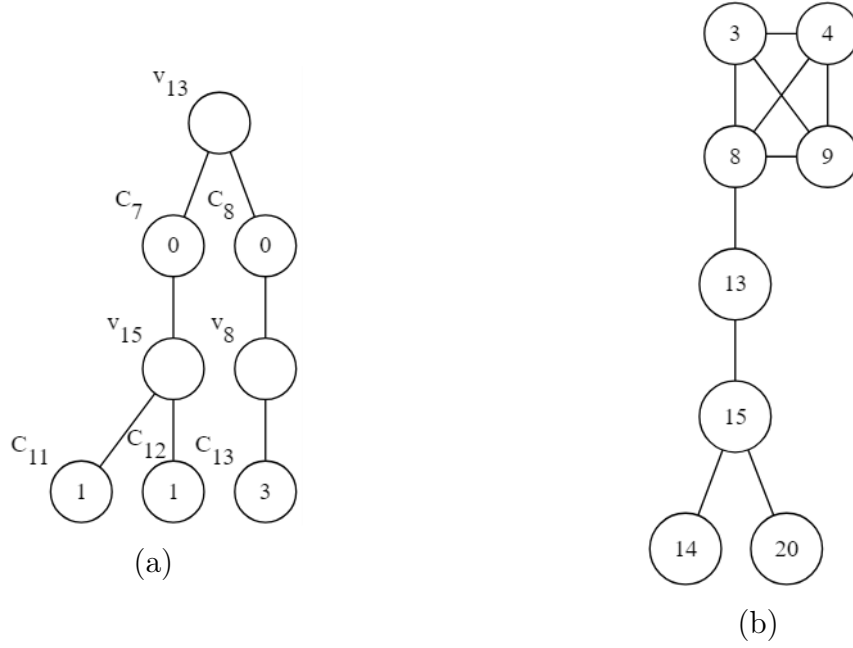


Figura 26 – A árvore auxiliar $T(B)_{v_{13}}^{v_{12}}$ e o subgrafo induzido $B_{v_{13}}^{v_{12}}$ do grafo bloco da Figura 25

Passo indutivo: $h > 2$, ou seja v é do tipo 2. Vamos mostrar a seguir que se a afirmação vale para alturas inferiores a h , ela vale para h , ou seja, para v . A hipótese de indução é que para as articulações u netas de v vale que $\beta(B_u^r) = \beta(B_u^r - u)$ se u é uma articulação candidata e que $\beta(B_u^r) = \beta(B_u^r - u) + 1$ se u não é uma articulação candidata.

Vamos considerar D_i o conjunto das articulações candidatas filhas de $s_i \in S$, ou seja, D_i nesse caso é $ac(T(B)_v^r, s_i)$ e F_i o conjunto das articulações que não são candidatas filhas de $s_i \in S$.

Vamos construir um emparelhamento máximo M para B_v^r . Inicialmente, para cada $s_i \in S$, vamos considerar M contendo $\bigcup_{j=1}^{|D_i|} M(B_{d_j}^r - d_j)$ e $\bigcup_{j=1}^{|F_i|} M(B_{f_j}^r)$. Com isso, pela hipótese de indução, para todo $s_i \in S$, todo $f \in F_i$ estará M -saturado pois precisa estar saturado em $M(B_f^r)$. Além disso todo $d \in D_i$ estará M -exposto e $M(B_d^r - d)$ é um emparelhamento máximo em B_d^r .

Observe que não existe caminho M -aumentante de nenhum vértice de B_d^r para $d \in D_i$, pois se houvesse, $M(B_d^r - d)$ não seria máximo. Também não existe caminho M -alternante de B_f^r para $f \in F_i$, pois se houvesse, f poderia estar exposto.

Além disso, emparelhe entre si os vértices da clique formada pelas articulações de D_i e os vértices simpliciais de cada $s_i \in S$. Haverá dois casos.

1. Se $|ac(T, s_i)| + peso(s_i)$ é par, então conseguiremos emparelhar todos os vértices correspondentes dois a dois. Portanto todos os vértices de s_i estão M -saturados.
2. Se $|ac(T, s_i)| + peso(s_i)$ é ímpar, então ao emparelhar dois a dois, sobrá exatamente um vértice, que é o único vértice M -exposto da clique

Vamos provar agora que não existe caminho M -aumentante em $B_v^r - v$. Vimos que não existe caminho M -aumentante nos grafos B_d^r e B_f^r para qualquer $d \in D_i$ e $f \in F_i$. Se s_i está no caso 1, então todos os vértices de s_i estão M -saturados e não existe caminho M -aumentante. Se s_i está no caso 2, existe algum vértice $d \in D_i$ ou vértice simplicial que está exposto.

Vamos agora considerar os dois casos para v , que ainda está M -exposto.

- Se v é articulação candidata, então todo $s_i \in S$ está no caso 1. Vamos provar que M é máximo em B_v^r . Vimos que não existe caminho M -aumentante em B_d^r e B_f^r para todo $d \in D_i$ e $f \in F_i$. Se s_i está no caso 1, então todos os vértices de s_i estão saturados e não existe caminho M -aumentante.

Portanto, M é máximo e v pode estar M -exposto. Em outras palavras, $\beta(B_v^r) = \beta(B_v^r - v)$ se v é uma articulação candidata.

- Se v não é articulação candidata, então algum $s_i \in S$ está no caso 2. Vamos provar que M não é máximo e v precisa estar M -saturado. Considere s_i uma das componentes que está no caso 2. Há nessa componente exatamente um vértice u M -exposto. Podemos então adicionar (v, u) a M e M agora será um emparelhamento máximo. Vamos provar que v precisa estar saturado pois não há caminho M -alternante de nenhum vértice para v .

Vimos que para todo $s_i \in S$ e para todo $d \in D_i$, mesmo d estando M -exposto, não existia caminho M -aumentante em $B_d^r - d$. Portanto não há M -alternante agora para v .

Também vimos que não existia M -alternante para $f \in F$ em B_f^r . Como um caminho M -alternante de algum vértice de B_f^r para v precisaria necessariamente passar por f , concluímos que não há M -alternante de nenhum vértice de B_f^r para v .

É possível também que algum vértice d ou um vértice simplicial tenha ficado M -exposto pois s_i está no caso 2. Não existe caminho M -alternante desse vértice até v pois esse caminho teria que passar por u e isso não é possível.

Portanto, M é máximo e v precisa estar M -saturado. Em outras palavras, $\beta(B_v^r) = \beta(B_v^r - v) + 1$ se v não é uma articulação candidata.

□

Corolário 1. v é uma articulação composta livre em um grafo bloco B se e somente se v é uma articulação candidata em $T(B)^v$ e uma articulação composta.

Demonstração. O Corolário 1 vale, pois trata-se de considerar v também uma articulação composta dos dois lados da proposição, a partir do Teorema 18. □

O Teorema 18, junto com Corolário 1, já abre margem para a criação de um algoritmo que determine as articulações compostas livres de B , caso existam o que equivale a dizer que assim pode-se determinar se a cardinalidade do emparelhamento desconexo é igual à do irrestrito ou não.

A partir de uma articulação composta v de um grafo bloco B , é possível determinar se v é uma articulação composta livre ou não com uma busca em profundidade simples em $T(B)^v$ determinando, recursivamente, as articulações candidatas. Se v for articulação candidata, então ela é uma articulação composta livre e $\beta_d(B) = \beta(B)$.

Nessa busca, determinam-se todas as articulações candidatas. Para saber aquelas que são articulações compostas livres bastaria, para cada articulação candidata u de $T(B)^v$, refazer a busca, iniciando-a por u . A partir disso, se u é articulação candidata em $T(B)^u$, então é articulação composta livre em B . Não é preciso analisar as articulações que não são candidatas w em $T(B)^v$, pois ser candidata em $T(B)^w$ é uma restrição maior que ser

candidata em $T(B)^v$. Isso acontece pois w em $T(B)^w$ tem exatamente um filho a mais do que tem em $T(B)^v$. Logo, se w não é articulação candidata em $T(B)^v$, então w também não é articulação candidata em $T(B)^v$. Esse algoritmo descrito teria complexidade de tempo $O(n^2)$.

No entanto, é possível construir um algoritmo ainda mais eficiente anunciado na sequência, em $O(n)$, no qual só é necessário fazer 2 buscas. Apresentaremos o Lema 7 a seguir, que será base para esse novo enfoque.

Lema 7. Sejam v e r duas articulações distintas em B , C o pai de v em $T(B)^r$ e v articulação candidata em $T(B)^r$. Então v é articulação candidata em $T(B)^v$ se e somente se $\text{peso}(C) + |\text{ac}(T(B)^v, C)|$ é par.

Demonstração. (\Rightarrow) :

Se v é articulação candidata em $T(B)^v$, então para todos os filhos u de v , incluindo C , vale que $\text{peso}(u) + |\text{ac}(T(B)^v, u)|$ é par. Logo, como uma condição mais fraca, v é articulação candidata em $T(B)^r$ também.

(\Leftarrow) :

Levando em conta que, ao enraizarmos $T(B)$ em v ao invés de r , v terá somente mais um filho imediato, C , para que v seja articulação candidata em $T(B)^v$ basta que $\text{peso}(C) + |\text{ac}(T(B)^v, C)|$ seja par, o que faz parte da condição. Então, v é articulação candidata em $T(B)^v$. \square

Apresentaremos a seguir um algoritmo recursivo que identifica todas as articulações compostas livres de um grafo bloco, com complexidade de tempo $O(n + m)$.

A ideia do algoritmo, quando existe articulação composta, é usar duas buscas em profundidade em $T(B)$. A primeira, a partir de um vértice qualquer v , originado de uma articulação de B , encontra todas as articulações candidatas de $T(B)^v$, percorrendo a árvore no sentido das folhas para a raiz, usando as propriedades do Teorema 18 e do Corolário 1.

Ao final da primeira busca, pode-se saber se v é articulação candidata em $T(B)^v$, ou seja, uma articulação composta livre em B . A partir dessa informação e do Lema 7, a segunda busca identifica as demais articulações compostas livres em B , processando os vértices de $T(B)^v$ no sentido da raiz para as folhas.

A seguir, descreveremos mais detalhadamente as funções utilizadas.

- A função $\text{AC}(B)$ recebe um grafo bloco B e retorna o conjunto de articulações compostas de B . O algoritmo pode utilizar uma classificação de cada vértice de B como articulação composta segundo o critério foi descrito anteriormente. Por esse motivo, o pseudocódigo não é mostrado.
- A função $\text{CARDEMPIRRESTRITOBLOCOS}(B)$ recebe um grafo bloco B e retorna a cardinalidade de um emparelhamento máximo irrestrito em B , ou seja, retorna $\beta(B)$. Essa função pode ser obtida a partir da implementação do Algoritmo 5.
- As funções $\text{MARCA}(v)$ e $\text{DESMARCA}(v)$ recebem um vértice v de $T(B)$ e, respectivamente, marcam e desmarcam v a partir de variáveis globais.
- A função $\text{ACL}(B)$ recebe um grafo bloco B e retorna o conjunto de articulações compostas livres em B .

- A função $BUSCA1(v)$ recebe um vértice de um grafo $T(B)$ originado de uma articulação em B e preenche parcialmente as listas de cada vértice de $T(B)$, o que determina as articulações candidatas de $T(B)^v$.
- A função $BUSCA2(v)$ recebe um vértice de um grafo $T(B)$ originado de uma articulação em B e preenche o restante das listas de cada vértice de $T(B)$, o que determina as articulações compostas livres de B .

Algoritmo 4 – Cardinalidade do Emparelhamento Desconexo máximo em um grafo bloco

```

1: função BUSCA1(Vértice:  $v$ , Inteiro:  $nivel$ )
2:   MARCA( $v$ )
3:   para  $u$  vizinho não marcado de  $v$  em  $T$  faça
4:     se BUSCA1( $u, nivel + 1$ ) então
5:        $lista[v] \leftarrow lista[v] \cup \{u\}$ 
6:   se  $nivel \bmod 2 = 1$  então
7:     retorna  $|lista[v]| = 0$ 
8:   senão
9:     retorna  $(|lista[v]| + PESO(v)) \bmod 2 = 1$ 
10:
11: função BUSCA2(Vértice:  $v$ , Inteiro:  $nivel$ )
12:   MARCA( $v$ )
13:   para  $u$  vizinho não marcado de  $v$  em  $T$  faça
14:     se  $nivel \bmod 2 = 1$  então
15:       se  $|lista[v] - \{u\}| = 0$  então
16:          $lista[u] \leftarrow lista[u] \cup \{v\}$ 
17:     senão
18:       se  $(|lista[v] - \{u\}| + PESO(v)) \bmod 2 = 1$  então
19:          $lista[u] \leftarrow lista[u] \cup \{v\}$ 
20:   BUSCA2( $u, nivel + 1$ )
21:
22: função ACL(Grafo Bloco:  $B$ )
23:    $T \leftarrow GRAFOAUXILIAR(B)$ 
24:    $v \leftarrow$  Vértice qualquer de  $T$  originado de uma articulação de  $B$ 
25:    $r \leftarrow \{\}$ 
26:   para  $u$  vértice de  $T$  faça
27:      $lista[u] \leftarrow \{\}$ 
28:     DESMARCA( $u$ )
29:   BUSCA1( $v, 1$ )
30:   para  $u$  vértice de  $T$  faça
31:     DESMARCA( $u$ )
32:   BUSCA2( $v, 1$ )
33:   para  $v$  articulação composta de  $B$  faça
34:     se  $|lista[v]| = 0$  então
35:        $r \leftarrow r \cup \{v\}$ 
36:   retorna  $r$ 
37:

```

```

38: #Dados: B: Grafo bloco
39: se  $|AC(B)| = 0$  então
40:    $\beta_d(B) = 1$  e  $M_d(B)$  é uma aresta qualquer
41: senão
42:    $\beta(B) \leftarrow \text{CARDEMPIRRESTRITOBLOCOS}(B)$ 
43:   se  $|ACL(B)| > 0$  então
44:      $\beta_d(B) = \beta(B)$  e  $M_d(B)$  pode ser qualquer  $M(B - v)$ ,
45:     para qualquer articulação composta livre  $v$ .
46:   senão
47:      $\beta_d(B) = \beta(B) - 1$  e  $M_d(B)$  pode ser qualquer  $M(B - v)$ ,
48:     para qualquer articulação composta  $v$ 

```

A seguir, apresentaremos um exemplo de execução do algoritmo em que usaremos o grafo bloco B descrito na Figura 25.

Como B possui articulação composta, teremos que chamar a função $ACL(B)$ para descobrir se existem articulações compostas livres e, se existirem, quais são elas. Em $ACL(B)$, escolhemos o vértice v_{12} como parâmetro para a chamada das funções *Busca1* e *Busca2*. Após a execução de cada uma das funções mencionadas, vamos mostrar como estão preenchidas as listas de cada vértice e as conclusões que podemos chegar.

Ao final de $BUSCA1(v_{12}, 1)$, teremos as listas segundo a Tabela 3.

Listas	Conteúdo	Listas	Conteúdo
$lista[C_9]$	$\{\}$	$lista[C_3]$	$\{\}$
$lista[v_{17}]$	$\{C_9\}$	$lista[C_{11}]$	$\{\}$
$lista[C_{10}]$	$\{\}$	$lista[C_{12}]$	$\{\}$
$lista[v_7]$	$\{\}$	$lista[v_{15}]$	$\{C_{11}, C_{12}\}$
$lista[C_5]$	$\{v_7\}$	$lista[C_7]$	$\{\}$
$lista[v_{11}]$	$\{\}$	$lista[C_{13}]$	$\{\}$
$lista[C_1]$	$\{v_{11}\}$	$lista[v_8]$	$\{C_{13}\}$
$lista[C_6]$	$\{\}$	$lista[C_8]$	$\{\}$
$lista[v_{19}]$	$\{\}$	$lista[v_{13}]$	$\{\}$
$lista[C_2]$	$\{v_{19}\}$	$lista[C_4]$	$\{v_{13}\}$
$lista[v_{12}]$	$\{C_2, C_3, C_4\}$		

Tabela 3 – Tabela com o conteúdo das listas após a execução de $BUSCA1(v_{12})$

Note que os vértices originados de articulações de B cujas respectivas listas estão vazias, ou seja $v_7, v_{11}, v_{13}, v_{19}$, são as articulações candidatas de $T(B)^{v_{12}}$. Nesse caso, como a raiz v_{12} não é uma articulação candidata em $T(B)^{v_{12}}$, já podemos concluir que v_{12} não é uma articulação composta livre em B .

Ao final de $BUSCA2(v_{12}, 1)$, teremos as listas segundo a Tabela 4.

Note que os vértices originados de articulações de B cujas listas estão vazias são v_{13} e v_{19} . Como ambos são articulações compostas, então v_{13} e v_{19} são as únicas articulações compostas livres em B .

Ao final do algoritmo, como existem duas articulações compostas livres, v_{13} e v_{19} , temos que $\beta_d(B) = \beta(B) = \beta(B - v_{13}) = \beta(B - v_{19})$. Portanto, $M_d(B)$ pode ser definido por $M(B - v_{13})$ ou $M(B - v_{19})$.

Observe um emparelhamento desconexo máximo em B na Figura 27. Note que, por coincidência, foi possível construir um emparelhamento sem que nenhum dos dois vértices

Listas	Conteúdo	Listas	Conteúdo
$lista[C_9]$	$\{v_{17}\}$	$lista[C_3]$	$\{\}$
$lista[v_{17}]$	$\{C_9\}$	$lista[C_{11}]$	$\{\}$
$lista[C_{10}]$	$\{\}$	$lista[C_{12}]$	$\{\}$
$lista[v_7]$	$\{C_5\}$	$lista[v_{15}]$	$\{C_{11}, C_{12}, C_7\}$
$lista[C_5]$	$\{v_7\}$	$lista[C_7]$	$\{v_{13}\}$
$lista[v_{11}]$	$\{C_1\}$	$lista[C_{13}]$	$\{\}$
$lista[C_1]$	$\{v_{11}\}$	$lista[v_8]$	$\{C_{13}, C_8\}$
$lista[C_6]$	$\{v_{19}\}$	$lista[C_8]$	$\{v_{13}\}$
$lista[v_{19}]$	$\{\}$	$lista[v_{13}]$	$\{\}$
$lista[C_2]$	$\{v_{19}\}$	$lista[C_4]$	$\{v_{13}\}$
$lista[v_{12}]$	$\{C_2, C_3, C_4\}$		

Tabela 4 – Tabela com o conteúdo das listas após a execução de $BUSCA2(v_{12})$

v_{13} e v_{19} estejam saturados.

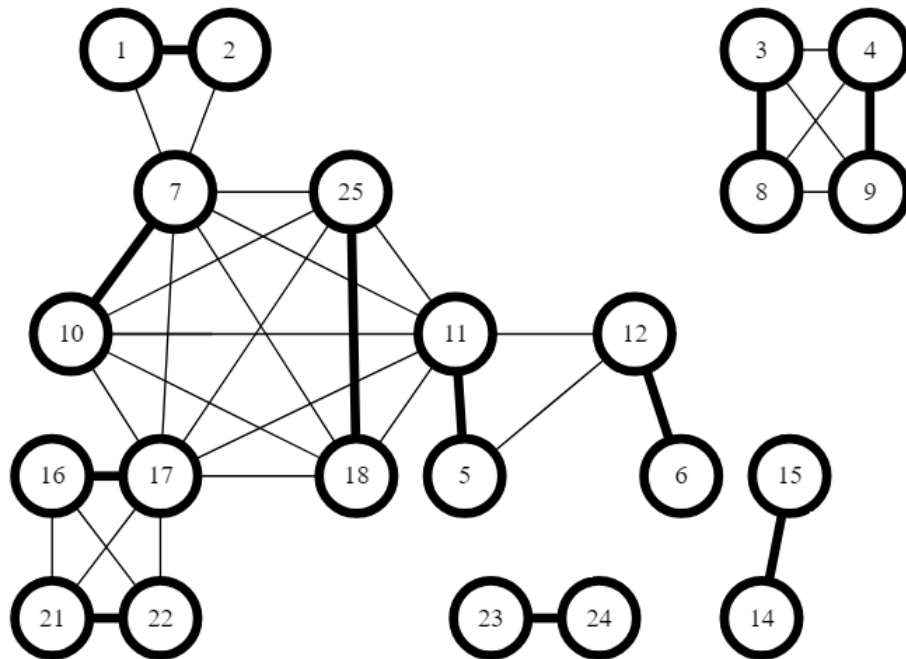


Figura 27 – Grafo induzido por um emparelhamento desconexo máximo em B

A seguir vamos demonstrar a corretude do Algoritmo 4 e a sua complexidade de tempo.

Teorema 19. O Algoritmo 4 identifica todas as articulações compostas livres em B .

Demonstração. O algoritmo recebe um grafo bloco B e, inicialmente, verifica se existe alguma articulação composta em B . Caso não exista, sabemos, pelo Teorema 16, que $\beta_d(B) = 1$. Caso exista, construímos $T(B)$ e escolhemos r articulação qualquer de B . O restante do algoritmo é dividido em duas buscas em profundidade a partir da raiz r , sendo uma para cada sentido de percurso na árvore. A primeira analisa os caminhos das folhas à raiz e, ao final, é capaz de identificar se r é articulação composta livre. A segunda analisa os caminhos da raiz às folhas e identifica as demais articulações compostas livres. Cada vértice v de $T(B)$ possui uma lista chamada de $lista[v]$. Para um vértice v , se v é

originado de uma articulação de B , então $lista[v]$ é conjunto de vértices filhos C tais que $|ac(T(B)^r, C)| + peso(C)$ é ímpar. Senão, $lista[v]$ é o conjunto de articulações candidatas filhas de v .

Durante a primeira busca, são identificadas em $T(B)^r$ as suas articulações candidatas da seguinte forma: se um nó v de $T(B)^r$, está em um nível par, então v representa um bloco em B . Neste caso, se $|ac(T(B)^r, v)| + peso(v)$ for ímpar, adiciona-se v à lista de seu pai. Se v está em nível ímpar, então representa uma articulação em B . Neste caso, v será adicionado à lista de seu pai se v for uma articulação candidata em $T(B)^r$. Ao final da busca, um vértice v relativo a uma articulação de B é uma articulação candidata em $T(B)^r$, se $|lista[v]| = 0$, ou seja, para todo filho C de v , vale que $|ac(T(B)^r, C)| + peso(C)$ é par. Isso se baseia no Corolário 1. Ao final da busca, sabemos se r é articulação candidata em $T(B)^r$ ou não. Se for articulação composta em B e candidata em $T(B)^r$, então é articulação composta livre em B . Resta analisar as outras articulações, o que é feito na próxima busca.

A segunda busca utiliza a ideia do Lema 7, e permite que se determine se cada articulação candidata assim permeneça, caso fosse a raiz da árvore. Isso pode ser feito, pois a primeira busca já coletou parte dessa informação. Percorrendo a árvore $T(B)^r$ no sentido da raiz para as folhas, quando estamos em um vértice v em nível ímpar, para cada filho C de v em $T(B)^r$, e nesse caso C representa um bloco em B , coloca-se v em $lista[C]$ se $|lista[v] - \{C\}| = 0$, pois se v passasse a ser filho de C , v seria uma articulação candidata. Quando estamos em vértice C em nível par, para cada filho v de C em $T(B)^r$, e nesse caso v representa uma articulação em B , coloca-se C em $lista[v]$ se $|lista[C] - \{v\}| + peso(C)$ é ímpar, pois se C passasse a ser filho de v , v não seria uma articulação candidata. Aplica-se recursivamente esse procedimento até as folhas.

Ao final da segunda busca, todos os vértices que representam uma articulação composta em B que possuem suas respectivas listas vazias são articulações compostas livres em B , segundo o Corolário 1. \square

Teorema 20. O Algoritmo 4 pode ser implementado em complexidade de tempo $O(n + m)$.

Demonstração. A função $CARDEMPÍRRESTRITOBLOCOS(B)$ pode ser implementada em $O(n + m)$. Essa prova foi omitida neste trabalho pelos motivos citados anteriormente. A geração do grafo auxiliar, que é uma árvore com $O(n)$ vértices, em $GRAFOAUXILIAR(B)$, também pode ser feita em $O(n + m)$ usando o algoritmo de decomposição em componentes biconexas (TARJAN, 1972). A mesma decomposição pode ser feita para que $AC(B)$ seja calculado em $O(n + m)$. As funções $BUSCA1(v)$ e $BUSCA2(v)$ são feitas em $O(n)$. Todas as manipulações de conjunto podem ser feitas a partir de listas de acesso direto com tamanho igual ao número de vértices do grafo auxiliar. Em $ACL(B)$, todos os loops têm complexidade $O(n)$, o que leva à conclusão que o algoritmo completo tem complexidade $O(n + m)$ também. \square

Vale observar que, a complexidade $O(n + m)$ é relativa à decomposição em componentes biconexas. Caso a entrada do programa já seja o grafo auxiliar que definimos, tanto a função $CARDEMPÍRRESTRITOBLOCOS(B)$ quanto o algoritmo possuirão complexidade de tempo $O(n')$, no qual n' é o número de vértices do grafo auxiliar.

3.5.1 Emparelhamento máximo em grafos blocos

A partir da teoria discutida nesta seção, apresentaremos, a seguir, um algoritmo recursivo que determina um emparelhamento máximo para um grafo bloco B , com complexidade de tempo $O(n + m)$.

O algoritmo usa uma única busca em profundidade muito similar à BUSCA1 do Algoritmo 4. Porém, ao invés de verificar a paridade dos vértices analisados, o Algoritmo 5 os emparelha.

A busca é feita em $T(B)$ a partir de um vértice r qualquer originado de uma articulação de B . O processamento dos vértices e a construção do emparelhamento são feitos no sentido das folhas à raiz de modo que, ao final da visita a um vértice v qualquer em $T(B)^r$, o algoritmo tenha calculado um emparelhamento parcial $M(B_v^r)$, que fará parte do emparelhamento máximo em B .

A seguir, descreveremos mais detalhadamente as funções utilizadas.

- A função $BUSCA(w, v, nivel)$ recebe dois vértices w e v de um grafo $T(B)^r$, e um inteiro correspondendo ao nível de v em $T(B)^r$. O processamento, feito no sentido das folhas à raiz, constrói $M(B_v^r)$, o adicionando à variável global M . A chamada inicial da função deve ser feita pela passagem de um mesmo vértice r nos dois parâmetros e 1, referente ao nível da raiz.
- A função $EMPARELHACLIQUE(K)$ recebe um conjunto de vértices K que induzem uma clique em B e os emparelha entre si em M . Caso $|K|$ seja ímpar, o vértice restante M -exposto é retornado. Senão, retorna um conjunto vazio.
- A função $EMPARELHAARTICULACAO(K, u)$ recebe um conjunto de vértices K de B e uma articulação u de B . Caso $|K| = 0$, então retorna-se u . Senão, há algum vértice w em K e então emparelhamos u com w em M e não retornamos vértices.

 Algoritmo 5 – Emparelhamento máximo em um grafo bloco

```

1: função BUSCA(Vértice:  $w$ , Vértice:  $v$ , Inteiro:  $nivel$ )
2:   para  $u$  vizinho de  $v$  em  $T$  faça
3:     se  $u \neq w$  então
4:       BUSCA( $v, u, nivel + 1$ )
5:   se  $nivel \bmod 2 = 1$  então
6:      $u \leftarrow$  Vértice de  $B$  representado em  $v$ 
7:      $lista[w] \leftarrow lista[w] \cup \text{EMPARELHAARTICULACAO}(lista[v], u)$ 
8:   senão
9:      $K \leftarrow$  Vértices de  $B$  representados em  $v$ 
10:     $lista[w] \leftarrow lista[w] \cup \text{EMPARELHACLIQUE}(K \cup lista[v])$ 
11:
12: função EMPARELHACLIQUE(Conjunto de vértices:  $K$ )
13:   enquanto existirem dois vértices  $u$  e  $w$  em  $K$   $M$ -expostos faça
14:      $M \leftarrow M \cup (u, w)$ 
15:   se existe algum vértice  $u$   $M$ -exposto então
16:     retorna  $\{u\}$ 
17:   senão
18:     retorna  $\{\}$ 
19:
20: função EMPARELHAARTICULACAO(Conjunto de vértices:  $K$ , Vértice:  $u$ )
21:   se  $|K| = 0$  então
22:     retorna  $\{u\}$ 
23:   senão
24:      $w \leftarrow$  Vértice qualquer de  $K$ 
25:      $M \leftarrow M \cup (u, w)$ 
26:     retorna  $\{\}$ 
27:
28: #Dados:  $B$ : Grafo bloco
29: #Global:  $M$ : Conjunto de arestas
30: #  $lista$ : Vetor de conjuntos de vértices
31:  $T \leftarrow \text{GRAFOAUXILIAR}(B)$ 
32:  $v \leftarrow$  Vértice qualquer de  $T$  originado de uma articulação de  $B$ 
33: para  $u \in V(T)$  faça
34:    $lista[u] \leftarrow \{\}$ 
35: BUSCA( $v, v, 1$ )
36:  $M(B) \leftarrow M$ 

```

A seguir, apresentaremos um exemplo de execução do algoritmo, onde usaremos o grafo bloco B descrito na Figura 25. Escolhemos o vértice v_{12} para iniciar a primeira chamada da busca. Considere que o percurso da busca em profundidade escolhido seja a sequência de vértices $v_{12}, C_1, v_{11}, C_5, v_{17}, C_9, v_7, C_{10}, C_2, v_{19}, C_6, C_3, C_4, v_{13}, C_7, v_{15}, C_{11}, C_{12}, C_8, v_8, C_{13}$.

Observe a Tabela 5, contendo todas as execuções de BUSCA($w, v, nivel$), a ordem em que elas são finalizadas e os emparelhamentos novos encontrados ao terminar cada função. Ao final, a variável M deverá conter todas as arestas encontradas, $\{(1, 2), (3, 8),$

$(5, 11), (6, 12), (7, 10), (4, 9), (14, 15), (16, 17), (18, 25), (21, 22), (23, 24)\}$. Observe que esse emparelhamento está representado na Figura 27.

Chamada	Arestas	Ordem	Chamada	Arestas	Ordem
BUSCA($v_{12}, v_{12}, 1$)	$\{(6, 12)\}$	21	BUSCA($v_{12}, C_3, 2$)	$\{\}$	11
BUSCA($v_{12}, C_1, 2$)	$\{(5, 11)\}$	7	BUSCA($v_{12}, C_4, 2$)	$\{\}$	20
BUSCA($C_1, v_{11}, 3$)	$\{\}$	6	BUSCA($C_4, v_{13}, 3$)	$\{\}$	19
BUSCA($v_{11}, C_5, 4$)	$\{(7, 10), (18, 25)\}$	5	BUSCA($v_{13}, C_7, 4$)	$\{\}$	15
BUSCA($C_5, v_{17}, 5$)	$\{(16, 17)\}$	2	BUSCA($C_7, v_{15}, 5$)	$\{(14, 15)\}$	14
BUSCA($v_{17}, C_9, 6$)	$\{(21, 22)\}$	1	BUSCA($v_{15}, C_{11}, 6$)	$\{\}$	12
BUSCA($C_5, v_7, 5$)	$\{\}$	4	BUSCA($v_{15}, C_{12}, 6$)	$\{\}$	13
BUSCA($v_7, C_{10}, 6$)	$\{(1, 2)\}$	3	BUSCA($v_{13}, C_8, 4$)	$\{\}$	18
BUSCA($v_{12}, C_2, 2$)	$\{\}$	10	BUSCA($C_8, v_8, 5$)	$\{(3, 8)\}$	17
BUSCA($C_2, v_{19}, 3$)	$\{\}$	9	BUSCA($v_8, C_{13}, 6$)	$\{(4, 9)\}$	16
BUSCA($v_{19}, C_6, 4$)	$\{(23, 24)\}$	8			

Tabela 5 – Tabela com o conteúdo de emparelhamentos novos encontrados após a execução da busca

A seguir vamos demonstrar a corretude do Algoritmo 5 e a sua complexidade de tempo.

Teorema 21. O Algoritmo 5 identifica $M(B)$ para um grafo bloco B .

Demonstração. O algoritmo inicia gerando a árvore $T(B)$ e começa a busca em um vértice v qualquer de $T(B)$.

Cada vértice u de $T(B)$ possui uma lista global chamada de $lista[u]$, inicializada como o conjunto vazio e que vai sendo preenchida após a busca de cada filho, contendo conjuntos de vértices conforme a explicação a seguir.

Se u é originado de uma articulação de B , então $lista[u]$ é conjunto de vértices de B construídos da seguinte forma. Se $M(B_C^v) = M(B_C^v - w)$, C filho de u em $T(B)^v$, w representado em C , então $lista[u]$ terá um único vértice qualquer representado por C . Ou seja, se é possível fazer um emparelhamento máximo M em B_C^v de modo que um vértice representado por C esteja M -exposto, então adiciona-se esse vértice à $lista[u]$.

Senão, u é originado de uma clique em B e $lista[u]$ corresponde ao conjunto de articulações candidatas filhas de u em $T(B)_u^v$.

Durante a BUSCA($w, u, nivel$), o algoritmo adiciona arestas ao emparelhamento global M de forma que $M(B_u^v) \subseteq M$ ao final dessa chamada. Essas arestas são determinadas da mesma forma que na indução do Teorema 18. Isto é, para um vértice u de $T(B)$, determinamos $M(B_u^v)$ da forma descrita a seguir. Considere $D = \{d_1, d_2, \dots, d_t\}$ e $F = \{f_1, f_2, \dots, f_r\}$ partições do conjunto de filhos de u em $T(B)^v$.

Se u é originado de uma articulação de B , C filho de u , então C pertence a D se e somente se $M(B_C^v) = M(B_C^v - z)$, z representado em C . Ou seja, se existe um vértice representado em C que pode estar M -exposto em um emparelhamento máximo $M(B_C^v)$, então $C \in D$. Caso contrário, $C \in F$. Vamos construir agora $M(B_u^v)$ a partir de um emparelhamento inicial M , contendo a união de $\bigcup_{i=1}^{|D|} M(B_{d_i}^v - z)$, z é um vértice qualquer de d_i e $\bigcup_{i=1}^{|F|} M(B_{f_i}^v)$. Considere K o conjunto dos vértices z mencionados. Para que tenhamos $M(B)_u^v$ em M , basta emparelhar entre si v com um vértice qualquer K . Se isso não for possível, ou seja, se K é vazio, então u pode estar exposto em $M(B)_u^v$.

Portanto, u é articulação candidata em $T(B)^v$ e, então, u é adicionado a lista de seu pai. Isso é feito na função $\text{EMPARELHAARTICULACAO}(K, u)$.

Se u é originado de uma clique de B , z filho de u , então z pertence a D se e somente se z é uma articulação candidata. O emparelhamento $M(B_u^v)$ também é construído a partir de um emparelhamento inicial M , contendo a união de $\bigcup_{i=1}^{|D|} M(B_{d_i}^v - d_i)$ com $\bigcup_{i=1}^{|F|} M(B_{f_i}^v)$. Considere K a união de D com os vértices simpliciais representados por u . Para que tenhamos $M(B_u^v)$ em M , basta emparelhar entre si o conjunto de vértices K . Depois disso, se $|K|$ é ímpar, adicionamos o vértice não emparelhado de K na lista do pai de u . Isso é feito na função $\text{EMPARELHACLIQUE}(K)$.

Portanto, ao final da busca, teremos calculado o emparelhamento máximo de B e associado à variável global M . \square

Teorema 22. O Algoritmo 5 pode ser implementado em complexidade de tempo $O(n + m)$.

Demonstração. O algoritmo inicialmente gera $T(B)$, que tem complexidade $O(n + m)$, e inicia as variáveis de lista para cada vértice de $T(B)$, que contém, no máximo, n vértices. A busca é executada para cada um desses vértices, ou seja, $O(n)$ vezes.

Para as funções $\text{EMPARELHACLIQUE}(K)$ e $\text{EMPARELHAARTICULACAO}(K, u)$, podemos observar que um vértice z de B só poderá aparecer contido em K , no máximo, uma vez em cada função. Também observamos que z só poderá aparecer contido em, no máximo, duas listas. Portanto, a execução de todas as operações de conjuntos e emparelhamentos do algoritmo é feita em $O(n)$.

Portanto, sendo a geração de $T(B)$ a operação mais custosa, o Algoritmo 5 tem complexidade de tempo $O(n + m)$. \square

3.6 Emparelhamento desconexo em grafos cordais

Nesta seção, apresentaremos resultados para a determinação de $\beta_d(G)$ e $M_d(G)$ para um grafo cordal qualquer G , de forma eficiente.

Sabemos que um grafo cordal G tem exatamente $n - 1$ separadores de vértices minimais e, portanto, no máximo essa mesma quantidade de separadores minimais. A partir do método para encontrar emparelhamentos desconexos máximos em grafos gerais, poderíamos remover cada separador minimal K e calcular o emparelhamento para o grafo $G - K$. O separador minimal que maximiza $\beta(G - K)$ determina um emparelhamento desconexo máximo, que seria $M(G - K)$. Utilizando o algoritmo de (MICALI; VAZIRANI, 1980) para calcular cada emparelhamento em $O(m\sqrt{n})$, a implementação completa para determinar um emparelhamento desconexo máximo teria complexidade de tempo $O(nm\sqrt{n})$.

Apresentaremos a seguir um algoritmo mais eficiente, com complexidade de tempo $O(nm)$, bem como sua base teórica e provas de corretude e de complexidade.

Primeiramente, vamos definir um separador composto, cuja ideia será usada para a determinação do emparelhamento desconexo.

Definição 6. Definimos um *separador composto* em um grafo cordal G como um separador cuja remoção cria, pelo menos, duas componentes conexas não triviais em G .

Vamos, então, considerar alguns resultados iniciais necessários para o desenvolvimento de um algoritmo que encontra um emparelhamento desconexo máximo em um grafo cordal.

Teorema 23. Seja G um grafo cordal. Então, $\beta_d(G) > 1$ se e somente se G possui um separador composto K .

Demonstração. (\Leftarrow) :

Se G possui um separador composto, então $G - K$ possui pelo menos duas componentes não triviais e, cada uma delas, gera um emparelhamento de tamanho pelo menos 1. Nesse caso, $\beta(G - K) > 1$ e, portanto, $\beta_d(G) > 1$.

(\Rightarrow) :

Se G não possui separador composto, então há duas possibilidades: G é trivial ou $\beta_d(G) = 1$. Nesse último caso não há duas arestas a_1 e a_2 em que os vértices incidentes de a_1 não sejam adjacentes a um dos vértices de a_2 . Portanto, $\beta_d(G) = 1$ por definição. \square

Teorema 24. Seja G um grafo cordal que possui separador composto e S o conjunto de separadores minimais de G . Então, $\beta(G - K) = \beta_d(G)$ e $M(G - K)$ é um emparelhamento desconexo máximo para algum $K \in S$.

Demonstração. Considere M um emparelhamento desconexo qualquer em G . Sabemos, pelo Teorema 23, que $|M| > 1$. Nesse caso, os vértices M -expostos formam um separador composto que, por sua vez, contém separadores minimais compostos. Dessa forma, basta considerar todos os emparelhamentos desconexos e cada separador minimal composto K e, então, calculamos $\beta(G - K)$. Considere K_{max} o separador que gera o maior valor para $\beta(G - K_{max})$. Esse valor é exatamente $\beta_d(G)$ e $M(G - K_{max})$ é um emparelhamento desconexo máximo de G . \square

Conforme visto na Seção 1.6, o conjunto de separadores minimais está contido no conjunto de separadores de vértices minimais. Para uma compatibilização com o algoritmo que será apresentado, podemos considerar o Teorema 24, de forma mais ampla, levando em conta o conjunto de separadores de vértices minimais ao invés de separadores minimais. Isso é dado no Corolário 2.

Corolário 2. Seja G um grafo cordal que possui separador composto e S o conjunto de separadores de vértices minimais de G . Então, $\beta(G - K) = \beta_d(G)$ e $M(G - K)$ é um emparelhamento desconexo máximo para algum $K \in S$.

Apresentaremos, a seguir, o Algoritmo 6, recursivo, que, a partir de uma clique tree T , relativa a um grafo cordal G , contendo, por definição, o conjunto de separadores de vértices minimais S , identifica os emparelhamentos máximos de $G - K$ para todo $K \in S$. Consequentemente, pelo Corolário 2, o algoritmo também identifica um emparelhamento desconexo máximo.

Inicialmente, o algoritmo identifica a ocorrência do caso simples do problema: se G não possui separadores compostos. Nesse caso, $\beta_d(G) = 1$ e $M_d(G)$ é uma aresta qualquer de G , segundo o Teorema 23.

Se G não satisfaz a esse caso, ou seja, G é não trivial e possui separador composto, precisamos fazer uma análise mais complexa. Para isso, usaremos duas buscas em profundidade em T , identificando emparelhamentos de subgrafos induzidos em G .

Esses emparelhamentos serão representados no algoritmo pelas variáveis globais $M_{a,b}$, que representam os emparelhamentos máximos no subgrafo induzido de G correspondente aos vértices representados na subárvore $T - (a,b)$ que contém a . Note que, ao retirar a aresta (a,b) de T , a clique tree será dividida em duas subárvores. Usaremos a notação A para representar a subárvore $T - (a,b)$ que contém o vértice a e B para representar a que contém o vértice b . Ao desconsiderar os vértices do separador (a,b) , A e B representam, cada uma, pelo menos, uma componente conexa em G . Então, para determinar $M_{a,b}$, vamos denotar S_{AB} como o conjunto dos vértices de G que estão representados em A e não

estão representados em B . Ou seja, basta considerarmos todos os vértices representados em A e retirarmos os vértices relativos ao separador (a,b) , pois estes estão em B . Assim, calculamos o emparelhamento máximo do grafo induzido em G pelos vértices de S_{AB} e guardamos na variável $M_{a,b}$.

Como exemplo, considere o grafo cordal G e a clique tree T descritos na Figura 28. Vamos mostrar os dados que envolvem a determinação de $M_{c,b}$. A subárvore C é composta pelos vértices c, e, f e B , por a, b, d, g . A aresta (c,b) de T representa o separador de vértices minimal $\{2, 5\}$ de G . O conjunto S_{CB} é igual a $\{4, 7, 9, 10\}$ e $M_{c,b}$ é um emparelhamento máximo do grafo induzido de S_{CB} em G . Nesse caso, só existe um emparelhamento máximo, $\{(4,9), (7,10)\}$.

Note que, se os dois grafos induzidos de um grafo cordal G pelos vértices de S_{AB} e de S_{BA} contêm pelo menos uma componente conexa não trivial, então (a,b) representa um separador composto. Além disso, podemos dizer que $M_{a,b} > 0$, $M_{b,a} > 0$ e, portanto, $M_{a,b} \cup M_{b,a}$ é um emparelhamento desconexo de G .

De acordo com o Corolário 2, $M(G - K)$ é um emparelhamento desconexo máximo para algum separador de vértices K , que está representado por alguma aresta (a,b) de T . Então, ao obtermos $M_{a,b}$ e $M_{b,a}$ para toda aresta (a,b) de T , conseguimos determinar um emparelhamento desconexo máximo. Basta encontrar uma aresta (a,b) que maximiza a soma $|M_{a,b}| + |M_{b,a}|$. Nesse caso, $M_d(G)$ seria $M_{a,b} \cup M_{b,a}$.

Para encontrar esses emparelhamentos do tipo $M_{a,b}$, o algoritmo usa duas buscas em profundidade a partir de um vértice v qualquer de T . A primeira busca processa as arestas no sentido das folhas à raiz de T^v e a segunda, no sentido da raiz às folhas.

Para que não seja necessário calcular os emparelhamentos integralmente a partir do grafo induzido, usamos emparelhamentos parciais e os aumentamos de maneira eficiente. Nesse caso, se estamos processando a aresta (a,b) , pela ordem de processamento das buscas descritas, para todos os vizinhos c de a , $c \neq b$, a aresta (a,c) de T já foi processada no sentido de c a a . Portanto, já possuímos o emparelhamento $M_{c,a}$. Sabemos que, ao retirar a aresta (a,b) de T , a árvore restante A possui todos os vértices de c representados, pois c está em A e a subárvore $A - (c,a)$ que contém c está contida em A .

Considere, então, $D_{a,b}$ como a diferença dos vértices de G que estão em a e não estão em b . Além disso, considere R como a união de todos os conjuntos de vértices S_{CA} , $c \neq b$, c vizinho de a . Note que se considerarmos o subgrafo para gerar $M_{a,b}$, ou seja, o subgrafo induzido de G pelos vértices de S_{AB} , vamos ter exatamente $|D_{a,b}|$ vértices a mais que em R . Portanto, se considerarmos M a união de todos os $M_{c,a}$, podemos concluir que o emparelhamento $M_{a,b}$ poderá ser maior em, no máximo, $|D_{a,b}|$ arestas, ou seja, $|M_{a,b}| \leq |D_{a,b}| + |M|$. Então, para chegar eficientemente ao emparelhamento $M_{a,b}$, vamos usar M como um emparelhamento inicial e aumentá-lo até que M seja máximo.

Sabemos que $D_{a,b}$ é uma clique em G . Podemos começar aumentando M de forma simples em $\lfloor |D_{a,b}|/2 \rfloor$ unidades ao emparelhar vértices dois a dois desse conjunto. Nesse momento, sabemos que M pode aumentar, no máximo, em $\lceil |D_{a,b}|/2 \rceil$ arestas até que M seja máximo. Então, vamos usar um algoritmo que aumenta um emparelhamento em uma unidade com complexidade de tempo $O(m)$ descrito em (MICALI; VAZIRANI, 1980) (VAZIRANI, 2012) (BLUM, 1990). Esse algoritmo é baseado na detecção de caminhos M -aumentantes. Quando chegamos ao emparelhamento M máximo, podemos atribuí-lo a $M_{a,b}$.

Para que calculemos os vértices usados nos grafos induzidos e os emparelhamentos parciais usados de forma eficiente, guardamos conjuntos de vértices acumulados e conjuntos de emparelhamentos acumulados da forma descrita a seguir. Para todos os conjuntos de

vértices S_{BA} utilizados para gerar $M_{b,a}$, b vizinho de a , adicionamos S_{BA} à variável Vac_a e adicionamos todas arestas de $M_{b,a}$ a Mac_a .

A seguir, descreveremos mais detalhadamente as funções utilizadas no algoritmo.

- A função $GERAEMPARELHAMENTODESCONEXO(T)$ é a função principal, que recebe uma clique tree, faz as chamadas às funções $BUSCA1$ e $BUSCA2$, processa os resultados e por fim determina e retorna um emparelhamento desconexo máximo encontrado.
- A função $EMPARELHACLIQUE(K)$ recebe um conjunto de vértices K cujo subgrafo induzido seja uma clique e os emparelha dois a dois. Ao final, esse emparelhamento é retornado.
- A função $EMPARELHARESTO(H, M)$ recebe um grafo H e um emparelhamento parcial M , que pode ser máximo ou não em H . A função executa um loop que encontra caminhos M -aumentantes e aumenta M em uma unidade a cada iteração. Ao final do loop, M é máximo e é retornado. A determinação de um caminho M -aumentante pode ser implementada em $O(m)$ como descrito em (MICALI; VAZIRANI, 1980) (VAZIRANI, 2012) (BLUM, 1990).
- A função $BUSCA1(w, v)$ recebe dois vértices de uma clique tree T referentes a um grafo cordal e, de forma recursiva, visita o restante dos vértices. O processamento, feito no sentido das folhas à raiz, preenche as variáveis $M_{v,w}$ e, parcialmente, as variáveis Vac_w , Mac_w , para w pai de v na busca. A chamada inicial da função deve ser feita pela passagem do mesmo vértice nos dois parâmetros.
- A função $BUSCA2(w, v)$ recebe dois vértices de uma clique tree referente a um grafo cordal e, de forma recursiva, visita o restante dos vértices. O processamento, feito no sentido da raiz às folhas, preenche as variáveis $M_{w,v}$, Vac_v , Mac_v , para w pai de v na busca. A chamada inicial da função deve ser feita pela passagem do mesmo vértice nos dois parâmetros.
- A função $DIFERENCA(v, w)$ recebe dois vértices v e w de uma clique tree e retorna os vértices que estão representados em v que não estão representados em w .
- A função $GRAFOINDUZIDO(G, S)$ recebe um grafo G e um conjunto de vértices S contidos em G e retorna o subgrafo induzido de S em G .

Algoritmo 6 – Emparelhamento desconexo máximo em um grafo cordal

```

1: função BUSCA2(Vértice:  $w$ , Vértice:  $u$ )
2:   se  $w \neq u$  então
3:      $D_{u,w} \leftarrow DIFERENCA(u, w)$ 
4:      $D_{w,u} \leftarrow DIFERENCA(w, u)$ 
5:      $H \leftarrow GRAFOINDUZIDO(G, Vac_w \cup D_{w,u} - Vac_u - D_{u,w})$ 
6:      $M_{w,u} \leftarrow EMPARELHARESTO(H, EMPARELHACLIQUE(D_{w,u}) \cup Mac_w - M_{u,w})$ 
7:      $Mac_u \leftarrow Mac_u \cup M_{w,u}$ 
8:      $Vac_u \leftarrow Vac_u \cup V(H)$ 
9:   para  $a$  vizinho de  $u$  faça
10:    se  $a \neq w$  então
11:      BUSCA2( $u, a$ )

```

```

12:
13: função EMPARELHARESTO(Grafo:  $H$ , Emparelhamento:  $M$ )
14:   enquanto existir caminho  $M$ -aumentante em  $H$  faça
15:      $P \leftarrow$  Caminho  $M$ -aumentante em  $H$ 
16:      $M \leftarrow M \oplus P$ 
17:   retorna  $M$ 
18:
19: função EMPARELHACLIQUE(Conjunto de vértices:  $K$ )
20:    $M_r \leftarrow \{\}$ 
21:   enquanto existirem dois vértices  $u$  e  $w$  em  $K$   $M_r$ -expostos faça
22:      $M_r \leftarrow M_r \cup (u, w)$ 
23:   retorna  $M_r$ 
24:
25: função BUSCA1(Vértice:  $w$ , Vértice:  $u$ )
26:   para  $a$  vizinho de  $u$  faça
27:     se  $a \neq w$  então
28:       BUSCA1( $u, a$ )
29:   se  $w \neq u$  então
30:      $D_{u,w} \leftarrow$  DIFERENCA( $u, w$ )
31:      $H \leftarrow$  GRAFOÍNDUZIDO( $G, Vac_u \cup D_{u,w}$ )
32:      $M_{v,w} \leftarrow$  EMPARELHARESTO( $H, EMPARELHACLIQUE(D_{u,w}) \cup Mac_u$ )
33:      $Mac_w \leftarrow Mac_w \cup M_{u,w}$ 
34:      $Vac_w \leftarrow Vac_w \cup V(H)$ 
35:
36: função GERAEMPARELHAMENTODESCONEXO(Clique Tree:  $T$ )
37:    $v \leftarrow$  Vértice qualquer de  $T$ 
38:   BUSCA1( $v, v$ )
39:   BUSCA2( $v, v$ )
40:    $\beta_d(G) \leftarrow 1$ 
41:    $M_d(G) \leftarrow$  aresta qualquer do grafo cordal representado por  $T$ 
42:   para  $(u, w)$  aresta de  $T$  faça
43:     se  $|M_{u,w} \cup M_{w,u}| > \beta_d(G)$  e  $|M_{u,w}| > 0$  e  $|M_{w,u}| > 0$  então
44:        $M_d(G) \leftarrow M_{u,w} \cup M_{w,u}$ 
45:        $\beta_d(G) \leftarrow |M_d(G)|$ 
46:   retorna  $M$ 
47:
48: #Dados:  $T$ : Clique Tree que representa um grafo cordal  $G$ 
49: #Global:
50: # $M_{u,v}$ : Emparelhamento,  $(u, v) \in E(T)$ 
51: # $Vac_v$ : Conjunto de vértices,  $v \in V(T)$ 
52: # $Mac_v$ : Emparelhamento,  $v \in V(T)$ 
53: para  $v$  em  $V(T)$  faça
54:    $Vac_v \leftarrow \{\}$ 
55:    $Mac_v \leftarrow \{\}$ 
56:  $M \leftarrow$  GERAEMPARELHAMENTODESCONEXO( $T$ )
57:  $\beta_d(G) \leftarrow |M|$ 

```

A seguir, apresentaremos um exemplo de execução do algoritmo usando o grafo cordal G e a clique tree T descritos na Figura 28.

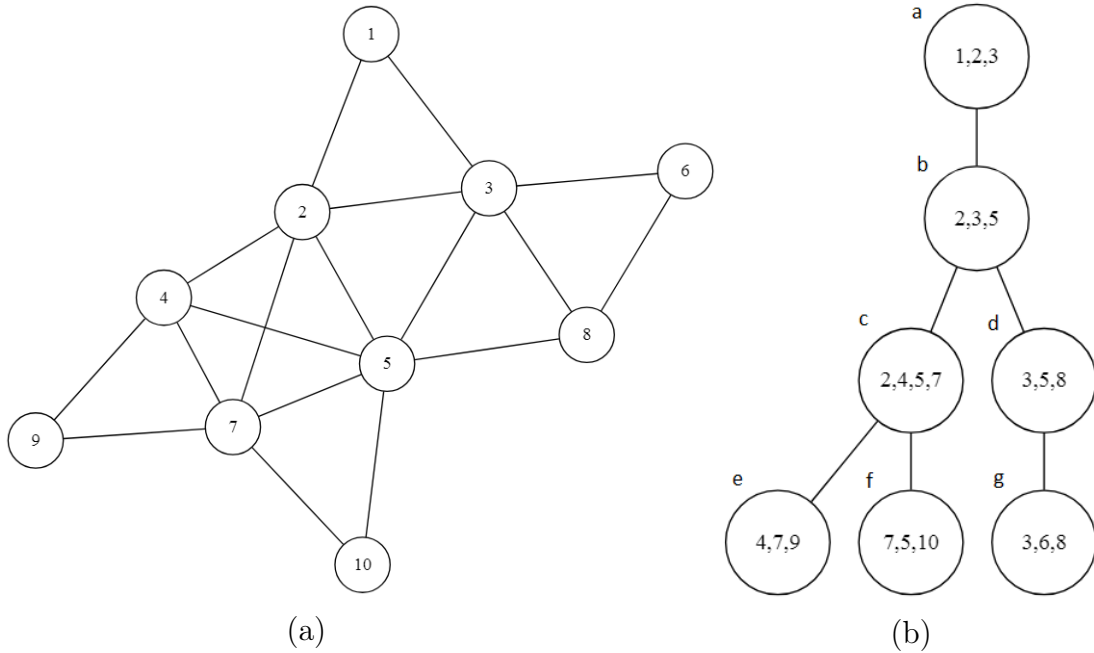


Figura 28 – Um grafo cordal e uma clique tree correspondente

Inicialmente, as variáveis Mac_v e Vac_v , $v \in V(T)$ recebem conjuntos vazios e é chamada a função $GERAEMPARELHAMENTODESCONEXO(T)$, que retornará um emparelhamento desconexo.

Quando executamos $GERAEMPARELHAMENTODESCONEXO(T)$, vamos atribuir a v o vértice a de T , o qual será usado para a primeira chamada das funções $BUSCA1(v,v)$ e $BUSCA2(v,v)$.

Na primeira busca, são preenchidas parcialmente as variáveis de emparelhamentos acumulados Mac_u e de vértices acumulados Vac_u , $u \in V(T)$, além dos emparelhamentos $M_{u,w}$, u filho de w em T^v . Ao final dessa busca, teremos os valores das variáveis conforme a Tabela 6 e a Tabela 7.

Variável	Conteúdo	Variável	Conteúdo
Vac_a	$\{4, 5, 6, 7, 8, 9, 10\}$	Mac_a	$\{(4,9), (7,10), (6,8)\}$
Vac_b	$\{4, 6, 7, 8, 9, 10\}$	Mac_b	$\{(4,9), (7,10), (6,8)\}$
Vac_c	$\{9, 10\}$	Mac_c	$\{\}$
Vac_d	$\{6\}$	Mac_d	$\{\}$
Vac_e	$\{\}$	Mac_e	$\{\}$
Vac_f	$\{\}$	Mac_f	$\{\}$
Vac_g	$\{\}$	Mac_g	$\{\}$

Tabela 6 – Tabela com o conteúdo das listas após a execução de $BUSCA1(a, a)$

Variável	Conteúdo	Variável	Conteúdo
$M_{a,b}$	$\{\}$	$M_{b,a}$	$\{(4,9), (7,10), (6,8)\}$
$M_{b,c}$	$\{\}$	$M_{c,b}$	$\{(4,9), (7,10)\}$
$M_{b,d}$	$\{\}$	$M_{d,b}$	$\{(6,8)\}$
$M_{c,e}$	$\{\}$	$M_{e,c}$	$\{\}$
$M_{c,f}$	$\{\}$	$M_{f,c}$	$\{\}$
$M_{d,g}$	$\{\}$	$M_{g,d}$	$\{\}$

Tabela 7 – Tabela com o conteúdo das listas após a execução de BUSCA1(a, a)

Na segunda busca, são preenchidas as variáveis de emparelhamentos acumulados Mac_u e de vértices acumulados Vac_u , $u \in V(T)$, além dos emparelhamentos $M_{u,w}$, u pai de w em T^v . Ao final dessa execução, teremos os valores das variáveis conforme a Tabela 8 e a Tabela 9.

Variável	Conteúdo	Variável	Conteúdo
Vac_a	$\{4, 5, 6, 7, 8, 9, 10\}$	Mac_a	$\{(4,9), (7,10), (6,8)\}$
Vac_b	$\{1, 4, 6, 7, 8, 9, 10\}$	Mac_b	$\{(4,9), (7,10), (6,8)\}$
Vac_c	$\{1, 3, 6, 8, 9, 10\}$	Mac_c	$\{(1,3), (6,8)\}$
Vac_d	$\{1, 2, 4, 7, 6, 9, 10\}$	Mac_d	$\{(4,9), (7,10), (1,2)\}$
Vac_e	$\{1, 2, 3, 5, 6, 8, 10\}$	Mac_e	$\{(1,3), (6,8), (2,5)\}$
Vac_f	$\{1, 2, 3, 4, 6, 8, 9\}$	Mac_f	$\{(1,3), (6,8), (2,4)\}$
Vac_g	$\{1, 2, 4, 5, 7, 9, 10\}$	Mac_g	$\{(4,9), (7,10), (1,2)\}$

Tabela 8 – Tabela com o conteúdo das listas após a execução de BUSCA2(a, a)

Variável	Conteúdo	Variável	Conteúdo
$M_{a,b}$	$\{\}$	$M_{b,a}$	$\{(4,9), (7,10), (6,8)\}$
$M_{b,c}$	$\{(1,3), (6,8)\}$	$M_{c,b}$	$\{(4,9), (7,10)\}$
$M_{b,d}$	$\{(4,9), (7,10), (1,2)\}$	$M_{d,b}$	$\{(6,8)\}$
$M_{c,e}$	$\{(1,3), (6,8), (2,5)\}$	$M_{e,c}$	$\{\}$
$M_{c,f}$	$\{(1,3), (6,8), (2,4)\}$	$M_{f,c}$	$\{\}$
$M_{d,g}$	$\{(4,9), (7,10), (1,2)\}$	$M_{g,d}$	$\{\}$

Tabela 9 – Tabela com o conteúdo das listas após a execução de BUSCA2(a, a)

Ao final das buscas, temos preenchidos todos os emparelhamentos $M_{v,w}$ e $M_{w,v}$, v vizinho de w em T . Nesse momento, todas as arestas de T são analisadas e o emparelhamento desconexo máximo corresponderá a $M_{v,w} \cup M_{w,v}$ para os vértices v e w que maximizam a soma $|M_{v,w}| + |M_{w,v}|$ tais que $|M_{v,w}| > 0$ e $|M_{w,v}| > 0$.

No exemplo mostrado, temos duas arestas e, conseqüentemente, dois separadores de vértices que geram emparelhamento desconexo máximo em G . A primeira aresta, (b,c) , representa o separador $\{2, 5\}$, cuja remoção gera o emparelhamento desconexo máximo $\{(1,3), (6,8), (4,9), (7,10)\}$, conforme a Figura 29a. A segunda, (b,d) , representa o separador $\{3, 5\}$, cuja remoção gera o emparelhamento desconexo máximo $\{(4,9), (7,10), (1,2), (6,8)\}$, conforme a Figura 29b.

A seguir vamos demonstrar a corretude do Algoritmo 6 e a sua complexidade de tempo.

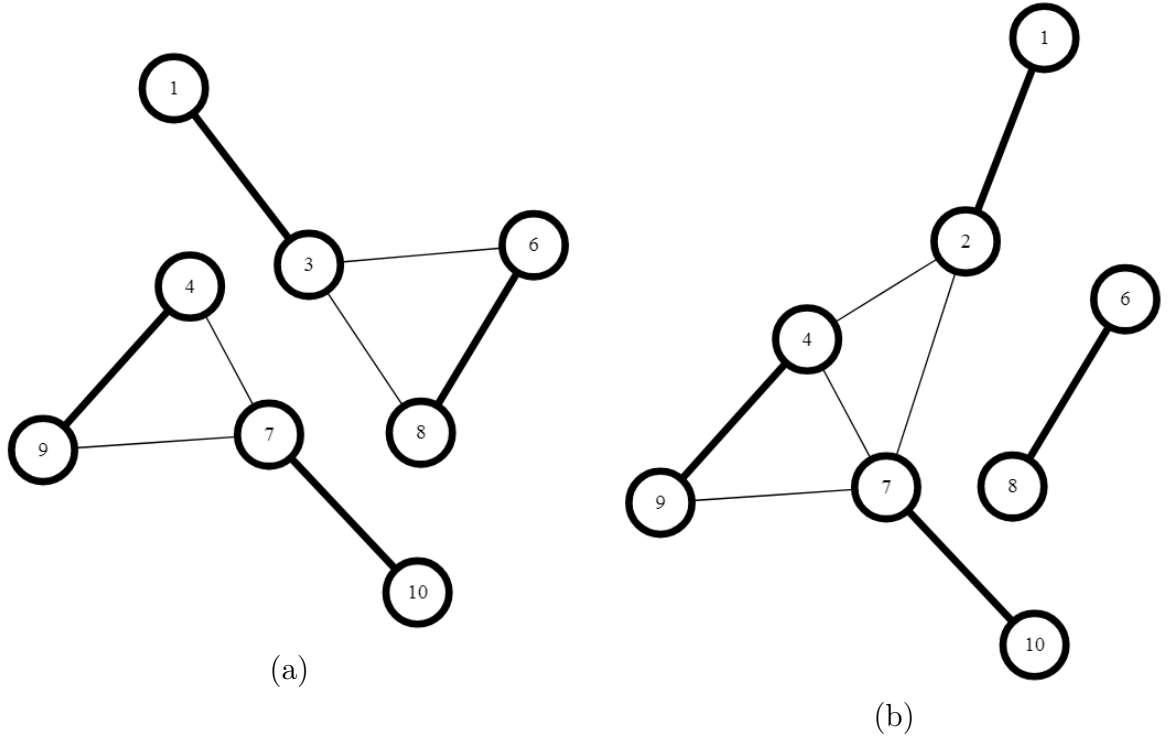


Figura 29 – Dois grafos induzidos por vértices de emparelhamentos desconexos máximos do grafo da Figura 28a

Teorema 25. O Algoritmo 6 identifica $M_d(G)$ e $M(G - K)$ para todo separador de vértices minimal K de um grafo cordal G .

Demonstração. Para a prova, vamos considerar T a clique tree de entrada do algoritmo que representa o grafo cordal G .

Inicialmente, o algoritmo inicializa as variáveis Vac e Mac e chama a função $GERAEMPARELHAMENTODESCONEXO(T)$, que retornará um emparelhamento desconexo máximo.

Um vértice v qualquer de T é escolhido para parâmetro da primeira chamada das buscas. A primeira busca determinará $M_{u,w}$ e a segunda, $M_{w,u}$, u filho de w em T^v .

Na função $BUSCA1(w,u)$, u é filho de w , com exceção da primeira chamada em que $u = w$ e que não alterará as variáveis globais. Portanto, para determinar o emparelhamento $M_{u,w}$, geramos um grafo H com os vértices S_{UW} de G que estão representados em T_u^v , com exceção dos vértices do separador K que está representado pela aresta (u,w) de T .

Pela recursão, para cada vértice u visitado, u filho de w , adicionamos os vértices S_{UW} na variável Vac_w . Portanto, se estamos processando um vértice u filho de w e queremos calcular S_{UW} , basta considerarmos o conjunto da união de Vac_u com os vértices de u que não estão em w e que estão sendo guardados na variável $D_{u,w}$.

Além disso, para todo vértice u visitado, u filho de w , adicionamos o emparelhamento $M_{u,w}$ à variável Mac_w . Portanto, se estamos em um vértice u filho de w e queremos calcular o emparelhamento $M_{u,w}$, podemos considerar como Mac_u um emparelhamento inicial e tentar aumentá-lo até que ele seja máximo. Observe que aumentamos o grafo induzido, pelo acréscimo dos vértices de $D_{u,w}$.

Tanto na $BUSCA1(w,u)$ quanto na $BUSCA2(w,u)$, para aumentar um emparelhamento M em um grafo H até que tenhamos o emparelhamento máximo $M_{u,w}$, vamos usar duas técnicas. Primeiro, vamos emparelhar todos os vértices que sabemos que estão expostos

e que formam uma clique. Esse emparelhamento é feito em $\text{EMPARELHACLIQUE}(K)$ e o adicionamos a M . Após isso, vamos aumentar achando caminhos M -aumentantes em H até que M seja máximo. Após essas duas etapas, temos $M_{u,w}$.

Na $\text{BUSCA2}(w,u)$, temos um caso parecido com a busca anterior, porém devemos agora considerar que as variáveis Vac e Mac já estão parcialmente preenchidas. Nessa busca, u é filho de w , com exceção da primeira chamada em que $u = w$ e que não alterará as variáveis globais. Para determinar o emparelhamento $M_{w,u}$, geramos um grafo H com os vértices S_{WU} de G , que estão representados em $T^w - T_u^w$, com exceção dos vértices do separador K que está representado pela aresta (u,w) de T .

Assim como na primeira busca, se calculamos o emparelhamento $M_{w,u}$, adicionamos S_{WU} à variável Vac_u e o emparelhamento $M_{w,u}$ à variável Mac_u . A diferença está em, a partir das variáveis que temos, como determinar S_{WU} e o emparelhamento inicial a ser aumentado.

Pela ordem de processamento dos vértices da segunda busca, se estamos calculando $M_{w,u}$, significa que, para todos os vizinhos a de u , $a \neq w$, Vac_u é exatamente a união de todos os S_{AU} e Mac_u é exatamente a união de todos os $M_{u,a}$. Além disso, Vac_w possui todos os vértices de $V(G)$, exceto os representados no próprio vértice w e Mac_w possui exatamente a união de todos os $M_{a,w}$, a vizinho de w .

Portanto, para obtermos S_{WU} , podemos considerar o conjunto Vac_w e retirar todos os vértices de T_u^w , ou seja, $Vac_u \cup D_{u,w}$. Também temos que adicionar os vértices de $D_{w,u}$, pois estes estão em w e não estão em u .

Para obtermos o emparelhamento M inicial, devemos considerar Mac_w e remover todas as arestas oriundas de T_u^w , ou seja, $M_{u,w}$. O aumento desse emparelhamento é feito da mesma maneira que na primeira busca.

Após a segunda busca, todas as variáveis do tipo $M_{u,w}$ estarão preenchidas e, então, podemos encontrar um emparelhamento desconexo máximo.

Usaremos o conceito que vimos no Corolário 2, considerando S o conjunto de separadores de vértices minimais de G . $M_d(G)$ pode ser obtido a partir de $M(G - K)$, para algum $K \in S$.

Sabemos também que, segundo o que foi apresentado em 1.6, cada aresta (a,b) de T representa um separador de vértices minimal em G correspondente à interseção dos vértices de G contidos em a e em b .

O loop da função $\text{GERAEMPARELHAMENTODESCONEXO}(T)$ percorre todas as arestas (u,w) de T , buscando o emparelhamento desconexo de maior cardinalidade. Como solução inicial, por sabermos que G tem arestas, consideramos o emparelhamento desconexo de uma aresta qualquer até que seja encontrada uma solução melhor. Precisamos, então, identificar quais outros emparelhamentos são candidatos para a resposta do problema.

Seja (u,w) uma aresta de T que representa um separador de vértices minimal K de G . Vamos mostrar que só precisamos analisar (u,w) se $|M_{u,w}| > 0$ e $|M_{w,u}| > 0$. Discutimos anteriormente que, se $|M_{u,w}| > 0$ e $|M_{w,u}| > 0$, então $M_{u,w} \cup M_{w,u}$ é um emparelhamento desconexo. Por outro lado, se $|M_{u,w}| = 0$ ou $|M_{w,u}| = 0$, então, o emparelhamento $M_{u,w} \cup M_{w,u}$ pode ser desconexo ou não. Em ambos os casos, vamos mostrar porque não é necessário considerar esse emparelhamento para encontrar um emparelhamento desconexo máximo no algoritmo.

Se $M_{u,w} \cup M_{w,u}$ não é um emparelhamento desconexo, então não é um emparelhamento desconexo máximo. Se $M_{u,w} \cup M_{w,u}$ é um emparelhamento desconexo e $M_{u,w} \cup M_{w,u}$ contém somente uma aresta, teremos nesse caso que, se $M_{u,w} \cup M_{w,u}$ for um emparelhamento desconexo máximo, então qualquer outra aresta também é um emparelhamento desco-

nexo máximo, o que é considerado como escolha inicial do algoritmo. Se $M_{u,w} \cup M_{w,u}$ é um emparelhamento desconexo e contém mais de uma aresta, há um separador minimal composto K_a contido nos vértices não saturados de G . Além disso, $K_a \neq K$, pois K não é um separador composto. Por isso, mesmo que $M_{u,w} \cup M_{w,u}$ seja um emparelhamento desconexo máximo, há um emparelhamento de cardinalidade maior ou igual ao remover o separador K_a . Esse separador corresponde a alguma aresta (a,b) de T , que será analisada no algoritmo e $M_{a,b} \cup M_{b,a}$ será um possível resultado para o mesmo, pois $|M_{a,b}| > 0$ e $|M_{b,a}| > 0$.

Ao final, portanto, teremos encontrado $M_d(G)$, que será o emparelhamento de maior cardinalidade que satisfaça as condições enunciadas anteriormente. Além disso, encontramos $M(G-K)$ para todo separador de vértices minimal K . O emparelhamento $M(G-K)$ pode ser obtido a partir das variáveis encontradas, pois, sabendo que K é representado por alguma aresta (u,w) de T , então $M(G-K)$ é $M_{u,w} \cup M_{w,u}$. \square

Teorema 26. O Algoritmo 6 pode ser implementado em complexidade de tempo $O(nm)$.

Demonstração. As operações de diferença, diferença simétrica, união e intersecção no conjunto de arestas ou de vértices dos grafos considerados podem ser feitas em $O(m)$, a partir de listas de acesso direto em que todos teriam, no máximo, m elementos.

A geração dos grafos induzidos é feita $O(n)$ vezes e pode ser implementada em $O(n+m)$. Portanto, a complexidade de tempo total, por busca, dessa geração, é $O(nm)$.

Cada uma das duas buscas percorre a clique tree T , passando uma vez por cada vértice. Como G possui no máximo n cliques maximais, então cada busca é executada $O(n)$ vezes.

Para os emparelhamentos, a função $\text{EMPARELHACLIQUE}(K)$ pode ser implementada em $O(|K|)$. A determinação de caminhos M -aumentantes em $\text{EMPARELHARESTO}(G, M)$ pode ser implementada em $O(m)$, como descrito em (MICALI; VAZIRANI, 1980) (VAZIRANI, 2012) (BLUM, 1990). Não vamos nos preocupar com quantas vezes a função é chamada e, sim, com a complexidade total para gerar todos os emparelhamentos de cada busca.

Cada vez que vamos calcular um emparelhamento $M_{u,w}$, temos um grafo H e um emparelhamento parcial M , que é um emparelhamento máximo de $H - D_{u,w}$. Nesse caso, $M_{u,w}$ pode ser maior que M em, no máximo, $|D_{u,w}|$ unidades. Ao emparelhamos $\lfloor |D_{u,w}|/2 \rfloor$ na primeira etapa, $\text{EMPARELHACLIQUE}(K)$, sobrarão exatamente $\lceil |D_{u,w}|/2 \rceil$ possíveis novas arestas para o emparelhamento $M_{u,w}$.

Note que, em cada busca, um vértice a de G aparecerá uma única vez em uma clique K a ser emparelhada por esse método.

Analisando a complexidade dessas operações, temos $|V(T)|$ é $O(n)$, ou seja, G possui $O(n)$ cliques maximais. Em um caso desfavorável, como, por exemplo, em um caminho, não conseguiríamos emparelhar nenhum vértice por $\text{EMPARELHACLIQUE}(K)$, pois o parâmetro K sempre seria um único vértice. Ou seja, para todos os pares de vértices vizinhos a e b da clique tree de um caminho, $D_{a,b} = 1$.

Então, pela função $\text{EMPARELHARESTO}(G, M)$, tentaríamos $O(n)$ vezes encontrar um caminho M -aumentante, o que resulta na complexidade de tempo total de $O(nm)$.

Ao final, o loop da função $\text{GERAEMPARELHAMENTODESCONEXO}(T)$ é executado exatamente $|V(T)| - 1$ vezes, que correspondem às arestas de T . Considerando a complexidade da união de conjuntos como $O(m)$, temos uma complexidade $O(nm)$ do loop.

A partir disso, podemos concluir que a complexidade de tempo do Algoritmo 6 é de $O(nm)$, sendo $n = |V(G)|$ e $m = |E(G)|$. \square

CONCLUSÕES E TRABALHOS FUTUROS

O tópico de emparelhamentos em grafos é antigo e amplamente estudado, pois leva a muitos resultados teóricos e práticos na Teoria de Grafos. Tradicionalmente, tem-se estudado emparelhamentos irrestritos com e sem pesos, os quais possuem vastos resultados na literatura.

Recentemente, surgiu o interesse pelo estudo de novos tipos de emparelhamentos em grafos, restritos a subgrafos, conforme, por exemplo, (GODDARD et al., 2005) e (CAMERON, 1989). Nesse estudo, são definidas propriedades dos subgrafos induzidos pelos vértices de um emparelhamento. Este trabalho tem foco no emparelhamento desconexo, no qual o subgrafo induzido é desconexo.

No Capítulo 1, são apresentados os conceitos e fundamentações teóricas de teoria da computação e teoria dos grafos usados no trabalho. São abordados temas como complexidade computacional, classes de problemas, definições e propriedades de grafos, definição e caracterizações de algumas classes de grafos, como árvores, grafos bloco, grafos cordais.

No Capítulo 2, são apresentados os problemas de emparelhamentos máximos e de emparelhamentos restritos a subgrafos. Detalhamos todos os conceitos envolvidos, como o de caminhos alternantes e aumentantes, detalhamos teoremas e alguns dos principais estudos da literatura sobre emparelhamentos. Além disso, foi feito um levantamento resumido de alguns dos estudos de emparelhamentos restritos a subgrafos recentes.

No Capítulo 3, aprofundamos o estudo nos emparelhamentos desconexos. Esse capítulo apresenta os resultados obtidos neste trabalho, que inclui algoritmos novos para resolver o problema. Além dos algoritmos, foi apresentada toda a base teórica que prova a corretude dos mesmos, que inclui definições, teoremas, corolários. Execuções dos algoritmos são exemplificados através de tabelas e figuras.

Os principais resultados alcançados foram os seguintes:

- O problema foi resolvido, de forma simples, para as seguintes classes: caminhos, ciclos, grafos completos, grafos split.
- Para árvores, o problema foi resolvido, tendo sido apresentado um algoritmo com complexidade de tempo $O(n)$.
- Para grafos bloco, foram apresentados dois algoritmos, ambos com complexidade de tempo $O(n + m)$, para resolver os problemas tanto de emparelhamento máximo quanto de emparelhamento desconexo máximo.
- Para grafos cordais, o problema foi resolvido e foi apresentado um algoritmo com complexidade de tempo $O(nm)$.

Os algoritmos para emparelhamentos desconexos máximos mencionados usaram uma técnica semelhante para resolver o problema. Essa abordagem usa uma representação do grafo em árvores enraizadas e faz uso de duas buscas em profundidade, de modo que a primeira processa os vértices das folhas à raiz e a segunda, da raiz às folhas. Além

disso, existe uma relação entre as representações e os separadores minimais dos grafos em questão.

Essas relações da representação podem ser observadas, por exemplo, em árvores, nas quais sabemos que os separadores minimais são os vértices internos. Então, usamos a representação original do grafo. Por outro lado, em grafos bloco, nos quais mostramos que os separadores minimais são as articulações que conectam os blocos, criamos uma representação em forma de árvore que, enraizada, relaciona os separadores minimais com a paridade dos níveis em que os vértices se encontram. Já em grafos cordais, usamos uma representação já conhecida, a clique tree, uma árvore da qual todos os separadores de vértices minimais estão representados em arestas e as cliques maximais, em vértices.

Os resultados deste trabalho estão sendo preparados para submissão ao IV Encontro de Teoria da Computação, a ser realizado no decorrer do CSBC 2019.

Trabalhos futuros

O problema de emparelhamentos restritos a subgrafos possui diversas questões em aberto para estudos teóricos novos. Podemos destacar a seguir algumas delas.

- Abordar emparelhamentos desconexos para grafos em geral, procurando desenvolver um algoritmo polinomial eficiente ou provar que esse problema pertence à classe NP-completo. Sendo esse tema ainda pouco estudado na literatura, a classe de problemas a que pertence a determinação de emparelhamentos desconexos para grafos gerais ainda permanece em aberto.

Há alguns resultados já obtidos que podem auxiliar nesse processo, como algoritmos que enumeram os separadores não triviais de um grafo geral (GOLDBERG, 1993) (KLOKS; KRATTSCH, 1998) (SHEN; LIANG, 1997) com complexidade $O(n^6 R_\Sigma)$, sendo R_Σ o número de separadores não triviais, o que pode ser exponencial.

- Estudar e desenvolver algoritmos para a determinação de emparelhamentos desconexos em outras classes de grafos, por exemplo, em grafos arco-circulares. Aparentemente, seria uma boa escolha para aprofundar os estudos, pois a classe possui uma vasta literatura, seu tratamento é mais complexo que grafos cordais, além de possuir diversas subclasses importantes, muito estudadas. Além disso, já é conhecido um algoritmo quadrático para determinação de separadores não triviais minimais para grafos arco-circulares (SUNDARAM; SINGH; RANGAN, 1994).
- Estudar o problema de emparelhamento desconexo para grafos com pesos. O problema de emparelhamentos com pesos, bastante conhecido e bem estudado, consiste em, a partir de um grafo e valores associados às arestas, determinar um conjunto de arestas que maximize a soma de seus valores e forme um emparelhamento.

O problema de emparelhamentos restritos a subgrafos com pesos ainda é pouco estudado, apesar de alguns resultados como, por exemplo, para emparelhamentos induzidos com pesos (KLEMZ; ROTE, 2017) (LIN; MESTRE; VASILIEV, 2018).

- Estudar uma variação do problema de emparelhamento desconexo no qual deseja-se que o subgrafo induzido pelo emparelhamento contenha um número específico de componentes conexas.

- Estudar outros tipos de emparelhamentos restritos a subgrafos como, por exemplo, emparelhamentos conexos, os quais podem ter relação com emparelhamentos desconexos e separadores. Alguns resultados desta dissertação podem servir de base para a resolução do problema em árvores, grafos bloco e grafos cordais. Além disso, o levantamento feito no Capítulo 2 evidencia que esse tipo de emparelhamentos ainda não tem muitos estudos relacionados.

REFERÊNCIAS

- BANDELT, H.-J.; MULDER, H. M. Distance-hereditary graphs. *Journal of Combinatorial Theory, Series B*, v. 41, n. 2, p. 182 – 208, 1986. ISSN 0095-8956.
- BERGE, C. Two theorems in graph theory. *Proceedings of the National Academy of Sciences*, National Academy of Sciences, v. 43, n. 9, p. 842–844, 1957. ISSN 0027-8424.
- BERRY, A.; SIMONET, G. Computing a clique tree with algorithm MLS (maximal label search). *CoRR*, abs/1610.09623, 2016.
- BLAIR, J.; PEYTON, B. An introduction to chordal graphs and clique trees. In: . [S.l.: s.n.], 1991.
- BLUM, N. *A New Approach to Maximum Matching in General Graphs*. [S.l.], 1990.
- BRANDSTÄDT, A.; HOÀNG, C. T. Maximum induced matchings for chordal graphs in linear time. *Algorithmica*, v. 52, n. 4, p. 440–447, Dec 2008. ISSN 1432-0541.
- BRANDSTÄDT, A.; LE, V.; SPINRAD, J. *Graph Classes: A Survey*. [S.l.]: Society for Industrial and Applied Mathematics, 1999.
- CAMERON, K. Induced matchings. *Discrete Applied Mathematics*, v. 24, n. 1, p. 97 – 102, 1989. ISSN 0166-218X.
- CAMERON, K. Connected matchings. Springer Berlin Heidelberg, Berlin, Heidelberg, p. 34–38, 2003.
- CHANDRAN, L. S. A linear time algorithm for enumerating all the minimum and minimal separators of a chordal graph. In: WANG, J. (Ed.). *Computing and Combinatorics*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001. p. 308–317. ISBN 978-3-540-44679-8.
- DIRAC, G. A. On rigid circuit graphs. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, v. 25, n. 1, p. 71–76, Apr 1961. ISSN 1865-8784.
- FRANCIS, M.; JACOB, D.; JANA, S. Uniquely restricted matchings in interval graphs. *SIAM Journal on Discrete Mathematics*, v. 32, n. 1, p. 148–172, 2018.
- FRICKE, G.; LASKAR, R. Strong matchings on trees. *Congr. Numer.*, v. 89, 01 1992.
- FUERST, M.; RAUTENBACH, D. On some hard and some tractable cases of the maximum acyclic matching problem. 10 2017.
- FULKERSON, D. R.; GROSS, O. A. Incidence matrices and interval graphs. *Pacific J. Math.*, Pacific Journal of Mathematics, A Non-profit Corporation, v. 15, n. 3, p. 835–855, 1965.
- GODDARD, W. et al. Generalized subgraph-restricted matchings in graphs. *Discrete Math.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 293, n. 1-3, p. 129–138, abr. 2005. ISSN 0012-365X.

- GOLDBERG, L. A. *Efficient Algorithms for Listing Combinatorial Structures*. [S.l.]: Cambridge University Press, 1993. (Distinguished Dissertations in Computer Science).
- GOLUMBIC, M. C.; LASKAR, R. C. Irredundancy in circular arc graphs. *Discrete Applied Mathematics*, v. 44, p. 79–89, 1993.
- GOLUMBIC, M. C.; LEWENSTEIN, M. New results on induced matchings. *Discrete Applied Mathematics*, v. 101, n. 1, p. 157 – 165, 2000. ISSN 0166-218X.
- HOPCROFT, J.; KARP, R. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, v. 2, n. 4, p. 225–231, 1973.
- HOWORKA, E. On metric properties of certain clique graphs. *Journal of Combinatorial Theory, Series B*, v. 27, n. 1, p. 67 – 74, 1979. ISSN 0095-8956.
- IBARRA, L. The clique-separator graph for chordal graphs. *Discrete Applied Mathematics*, v. 157, n. 8, p. 1737 – 1749, 2009. ISSN 0166-218X.
- KLEMZ, B.; ROTE, G. Linear-time algorithms for maximum-weight induced matchings and minimum chain covers in convex bipartite graphs. *CoRR*, abs/1711.04496, 2017.
- KLOKS, T.; KRATSCH, D. Listing all minimal separators of a graph. *SIAM Journal on Computing*, v. 27, n. 3, p. 605–613, 1998.
- LEWIS, J. G.; PEYTON, B. W.; POTHEN, A. A fast algorithm for reordering sparse matrices for parallel factorization. *SIAM J. Sci. Stat. Comput.*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, v. 10, n. 6, p. 1146–1173, nov. 1989. ISSN 0196-5204.
- LIN, M. C.; MESTRE, J.; VASILIEV, S. Approximating weighted induced matchings. *Discrete Applied Mathematics*, v. 243, p. 304 – 310, 2018. ISSN 0166-218X.
- MICALI, S.; VAZIRANI, V. V. An $o(v|v|c|e|)$ algorithm for finding maximum matching in general graphs. In: *21st Annual Symposium on Foundations of Computer Science (sfcs 1980)*. [S.l.: s.n.], 1980. p. 17–27. ISSN 0272-5428.
- PANDA, B. S.; PRADHAN, D. Acyclic matchings in subclasses of bipartite graphs. *Discrete Mathematics, Algorithms and Applications*, v. 04, n. 04, p. 1250050, 2012.
- ROSE, D.; TARJAN, R.; LUEKER, G. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing*, v. 5, n. 2, p. 266–283, 1976.
- SAVAGE, C. Maximum matchings and trees. *Information Processing Letters*, v. 10, n. 4, p. 202 – 205, 1980. ISSN 0020-0190.
- SHEN, H.; LIANG, W. Efficient enumeration of all minimal separators in a graph. *Theoretical Computer Science*, v. 180, n. 1, p. 169 – 180, 1997. ISSN 0304-3975.
- SPINRAD, J. Efficient graph representations. v. 19, p. 258, 01 2003.
- SUNDARAM, R.; SINGH, K. S.; RANGAN, C. Treewidth of circular-arc graphs. *SIAM J. Discrete Math.*, v. 7, p. 647–655, 11 1994.

SZWARCFITER, J. L. *Teoria Computacional dos Grafos: Os Algoritmos*. 1. ed. Amsterdam: Elsevier, 2018.

SZWARCFITER, J. L.; MARKENZON, L. *Estruturas de Dados e seus Algoritmos*. 3. ed. Rio de Janeiro, RJ: LTC Editora, 2010.

TARJAN, R. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, v. 1, n. 2, p. 146–160, 1972.

TARJAN, R. E.; YANNAKAKIS, M. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, v. 13, n. 3, p. 566–579, jul. 1984. ISSN 0097-5397.

VAZIRANI, V. V. An improved definition of blossoms and a simpler proof of the MV matching algorithm. *CoRR*, abs/1210.4594, 2012.

VUŠKOVIĆ, K. Even-hole-free graphs: A survey. *Applicable Analysis and Discrete Mathematics*, University of Belgrade, Serbia, v. 4, n. 2, p. 219–240, 2010. ISSN 14528630, 2406100X.