UNIVERSIDADE FEDERAL FLUMINENSE

MÍRIAM DE SOUZA MONTEIRO

Um estudo da técnica de decomposição em árvore na resolução de problemas combinatórios

NITERÓI

2009

UNIVERSIDADE FEDERAL FLUMINENSE

MÍRIAM DE SOUZA MONTEIRO

Um estudo da técnica de decomposição em árvore na resolução de problemas combinatórios

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre. Área de concentração: Otimização Combinatória.

Orientador:
Loana Tito Nogueira

NITERÓI

2009

Um estudo da técnica de decomposição em árvore na resolução de problemas combinatórios

Míriam de Souza Monteiro

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre.

Aprovada por:

Profª. D.Sc. Loana Tito Nogueira / IC-UFF (Presidente)

Profª. D.Sc. Sulamita Klein / UFRJ

Prof. D.Sc. Luiz Satoru Ochi/ UFF

Niterói, 12 de fevereiro de 2009.

Agradecimentos

Obrigada ao Senhor Deus, por te me protegido em todas as minhas viagens e por ter colocado em minha vida pessoas muito especiais.

Nem todas as palavras mais belas do mundo, nem todos os "obrigadas" possíveis e nem toda a gratidão existente serão suficientes para agradecer aos dois anjos que ajudaram na realização desse sonho: Héliton e Loana.

Héliton, meu marido, meu amor, meu companheiro - obrigada por acreditar em mim, financiando esse projeto e sendo meu alicerce de carinho em todos os momentos.

Loana, um anjo em forma de orientadora. Obrigada, obrigada, obrigada ... Obrigada por me dar um pouco do seu tempo, sua paciência, sua dedicação e por toda a ajuda dada nas provas dos teoremas mais loucos.

E o que seria de mim sem meus sobrinhos Larissa e Nathan. Lala e Tan obrigada por entenderem que a tia Mirinha não podia estar com vocês e mesmo assim não deixarem de me dar o amor mais puro que o ser humano pode receber.

Resumo

Muitos problemas reais podem ser modelados através de grafos. No entanto, esses problemas são geralmente NP-difícies e, portanto, não existem algoritmos eficientes para resolvê-los, a menos que $P = NP$.

Uma maneira de tentar "superar" tal dificuldade é resolver o problema quando restrito a classes especiais de grafos. Por exemplo, vários problemas NP-completos podem ser resolvidos eficientemente quando restritos às árvores. Uma questão bastante estudada atualmente é saber o quanto um grafo arbitrário se assemelha a uma árvore, motivado pela esperança de poder encontrar algoritmos eficientes que explorem a estrutura "tipo-árvore" desses grafos. Motivado por este fato, Robertson e Seymour introduziram a noção de lagura em árvore de um grafo e a noção relativa de decomposição em árvore.

Nesse trabalho fazemos um estudo da técnica de decomposição em árvore para grafos em geral. Iniciamos fazendo uma introdução ao conceito de decomposição em árvore e suas principais características. Apresentamos algoritmos que constróem decomposições em árvore para certas classes de grafos e utilizamos essas decomposições para construir soluções para os problemas de conjunto independente ponderado e coloração de grafos, que são, conhecidamente, NP-Difíceis. Ao final, apresentamos a complexidade algoritmica na construção de decomposições em árvore.

Palavras-chave: Decomposição em árvore, largura em árvore, k-árvore.

Abstract

Many real problems can be modeled by graphs. Many of these problems are generally NP-hard and, therefore, it does not exist polynomial time algorithms to solve them efficiently, unless $P = NP$.

A way of trying to overcome such difficulty is to solve the problem when restricted to special classes of graphs. For example, many NP-complete problems can be solved efficiently when restricted to trees. A very well studied question is to know how much a graph resembles a tree, motivated by the hope of finding efficient algorithms that exploit the tree-like structure of such graphs, Robertson and Seymour introduced the notion of treewidth and tree decomposition.

In this work, we study the treewidth technique for general graphs. We begin by introducing the concepts of tree decomposition and its major characteristics. We present algorithms the tree decomposition for certain classes of graphs and use these decompositions to construct the solution for the weighted independent set problem and for the coloring problem, which are known to be NP-hard problems. In the end, we show the complexity of constructing the tree decomposition of a graph.

Keywords: treewidth, width, tree decomposition, k-tree.

Palavras-chave

1. Decomposição em árvore
2. Largura em árvore
3. k-árvore

Sumário

Lista de Figuras **ix**

Lista de Tabelas **xi**

1 Introdução **1**

1.1 Grafos - Definições e Notações 2

1.2 Complexidade de Algoritmos 4

2 Decomposição em árvore **7**

2.0.1 O problema de decomposição em árvore 9

2.1 Decomposição em árvore enraizada 10

2.2 Algumas propriedades de DEA 11

2.2.1 DEA para algumas classes especiais de grafos 14

3 Largura em árvore de classes especiais de grafos **18**

3.1 Grafos cordais 18

3.1.1 Algumas definições 19

3.1.2 Decomposição em árvore da classe dos grafos cordais 23

3.1.3 Decomposição em árvore de Grafos Cordais - Algoritmo 26

3.2 K-árvores 27

3.2.1 Caracterizações de k-árvores 28

3.2.2 Lemas, Teoremas e Corolários 29

3.2.3 Algoritmo de decomposição em árvore para k-árvores 34

3.3 Heurísticas de triangularização 36
3.3.1 Lemas, Teoremas e Corolários 36
3.3.2 Exemplo de triangularização 39
3.4 Boa decomposição em árvore 41

4 Como resolver problemas NP-Difíceis com a técnica de DEA 47

4.1 Problema do conjunto independente ponderado 47
4.2 Problema do conjunto independente ponderado em árvores 48
4.3 Problema do conjunto independente ponderado em DEA 50
4.3.1 Construindo a tabela para um nó folha 50
4.3.2 Construindo a tabela para um nó introduz 50
4.3.3 Construindo a tabela para um nó esquece 51
4.3.4 Construindo a tabela para um nó junção 52
4.3.5 Construindo a solução com a utilização de todas as tabelas 53
4.4 Problema de coloração por lista 58
4.4.1 Coloração com listas 58
4.4.2 Construindo a solução 59

5 Conclusão 63

5.1 Trabalhos Futuros 64

Apêndice A – Complexidade em encontrar decomposições em árvore 65

A.1 Introdução 65
A.2 Algoritmo de tempo linear para o problema largura em árvore k 66
A.2.1 Preliminares 66
A.2.2 Algoritmo principal 73
A.3 NP-Completude do problema largura em árvore 76
A.3.1 Definições 76

A.3.2 Lemas, Teoremas e Corolários 81

Referências **87**

Lista de Figuras

2.1 Exemplos de DEA 9

2.2 Grafo, Subgrafo, DEA e subDEA 11

2.3 Decomposição em árvore de um grafo usando suas componentes 12

2.4 Subárvores que satisfazem a Propriedade de Helly 13

2.5 Uma ilustração da prova do Lema 2.3 17

3.1 (a)Grafo cordal; (b) Grafo não-cordal; (c) Conjunto adjacência do vértice simplicial a; (d) S é um ac-Separador; (e) S' é um ac-Separador Minimal . 19

3.2 (a) O grafo G separado em duas componentes A e B pelo separador S; (b) Os caminhos P_1 e P_2 do grafo G; (c) As cordas proibidas pela minimalidade de r e t 20

3.3 Ciclo C da prova do Lema 3.1 20

3.4 Ciclo C da prova do Lema 3.2 21

3.5 Decomposições $D'_j s$ das componentes de G unidas pelo vértice i^* 24

3.6 (a) Vértices simpliciais e esquema de eliminação perfeito ; (b) Triangularização 26

3.7 Decomposição em árvore de largura mínima de um grafo cordal 28

3.8 (a) 4-árvore com 4 vértices; (b) 4-árvore com 5 vértices; (c) 4-árvore com 6 vértices; (d) 4-árvore com 8 vértices 29

3.9 (a) Grafo a ser imerso em uma 3-árvore; (b) Grafo já imerso em uma 3-árvore 31

3.10 (a) Uma 3-árvore parcial G imersa em uma 3-tree H; (b) Sua árvore-EEP P; (c) Vértices de P 35

3.11 Grafo G a ser triangularizado 39

3.12 Uma execução do algoritmo Lex_M 44

3.13 Decomposição do grafo para a construção da boa-DEA 45

3.14 Construindo uma boa-DEA 46

4.1 Exemplo de conjunto independente e conjunto independente ponderado . . 48

4.2 Exemplo da construção de solução do PCI para uma árvore 49

4.3 Boa-DEA e a numeração de seus nós 53

4.4 Grafo a ser colorido e listas de restrição 62

A.1 Exemplos de grafo parcial (subgrafo gerador), subgrafo e supergrafo 76

A.2 Exemplo de um grafo k-decomponível 77

A.3 Arestas de preenchimento e grafo preenchido com respeito a π 78

A.4 Caminho A-cordal e grafo caminho-cordal 78

A.5 Exemplo de blocos de um grafo G e de um esquema de eliminação bloco-contínuo 79

A.6 Exemplo de uma ordenação em cadeia e de um grafo cadeia 79

A.7 Exemplo de um grafo $C(G)$ 80

A.8 Valor de corte para uma determinada permutação τ 80

A.9 Exemplo de construção do grafo bipartido $G'(A \cup B, E')$ 81

A.10 Um grafo $C(G')$ preenchido 85

A.11 Adjacências de um vértice A_i no momento de sua eliminação 86

Lista de Tabelas

2.1 Algumas classes de grafos com largura em árvore limitada 10

Capítulo 1

Introdução

Muitos problemas reais podem ser modelados através de grafos. No entanto, esses problemas são geralmente NP-difícies e, portanto, não existem algoritmos eficientes para resolvê-los, a menos que $P = NP$.

Uma maneira tentar "evitar" tal dificuldade é resolver o problema quando restrito a classes especiais de grafos. Por exemplo, vários problemas NP-completos podem ser resolvidos eficientemente quando restritos às árvores. Uma questão bastante estudada atualmente é saber o quanto um grafo arbitrário se assemelha a uma árvore, motivado pela esperança de poder encontrar algoritmos eficientes que explorem a estrutura "tipo-árvore" desses grafos. Motivado por este fato, Robertson e Seymour [23] introduziram a noção de *largura em árvore* de um grafo e a noção relativa de *decomposição em árvore*. A decomposição em árvore para um grafo procura estabelecer uma relação entre este grafo e uma árvore. Quanto menor a largura em árvore de um grafo, mais semelhante este é a uma árvore.

O objetivo principal deste trabalho consiste em procurar apresentar um estudo aprofundado e detalhado da técnica de decomposição em árvore, mostrando como ela pode ser utilizada na resoluçao de problemas combinatórios. Para isso, o texto foi escrito da maneira mais prática e simples possível de modo a se tornar um bom material para estudos futuros do assunto.

Esta dissertação está organizada como segue. No Capítulo 1 definimos os principais conceitos, introduzimos a notação utilizada e a noção de classes de complexidade. No Capítulo 2 apresentamos o conceito formal de decomposição em árvore, mostrando suas principais propriedades. No Capítulo 3 são apresentadas algumas classes de grafos com largura limitada. No Capítulo 4 mostramos exemplos de problemas combinatoriais que

podem ser resolvidos utilizando a técnica de decomposição em árvore e como resolvê-los. Finalmente, no Capítulo 5 fornecemos conclusões deste trabalho e algumas propostas de trabalho futuros. Pode-se ainda encontrar neste trabalho (Apêndice A) a prova de que determinar a largura em árvore de um grafo é NP-Completo e um algoritmo linear que testa se um dado grafo G tem largura em árvore no máximo k, k constante, e, em caso afirmativo, retorne uma decomposição em árvore de G com largura em árvore no máximo k. Neste trabalho optamos pela didática, inclusive na divisão dos capítulos. Assim, esta dissertação está organizada como segue. No Capítulo 1 definimos os principais conceitos, introduzimos a notação utilizada e a noção de classes de complexidade. No Capítulo 2 apresentamos o conceito formal de decomposição em árvore, mostrando suas principais propriedades. No Capítulo 3 são apresentadas algumas classes de grafos com largura limitada. No Capítulo 4 mostramos exemplos de problemas combinatoriais que podem ser resolvidos utilizando a técnica de decomposição em árvore e como resolvê-los. Finalmente, no Capítulo 5 fornecemos conclusões deste trabalho e algumas propostas de trabalho futuros. Pode-se ainda encontrar neste trabalho (Apêndice A) a prova de que determinar a largura em árvore de um grafo é NP-Completo e um algoritmo linear que testa se um dado grafo G tem largura em árvore no máximo k, k constante, e, em caso afirmativo, retorne uma decomposição em árvore de G com largura em árvore no máximo k.

A seguir, estabeleceremos algumas definições e notações que serão utilizadas ao longo deste trabalho.

1.1 Grafos - Definições e Notações

Um *grafo simples* é um par ordenado $G = (V, E)$, onde V é um conjunto finito não-vazio de *vértices*, denotado por $V(G)$ e E é um conjunto de pares não-ordenados de vértices distintos, chamados *arestas*, e denotado por $E(G)$. Utilizaremos a notação $n = |V(G)|$ e $m = |E(G)|$ para denotarmos a cardinalidade de $V(G)$ e $E(G)$, respectivamente. Definimos as arestas como sendo pares não-ordenados, mas serão indicadas como sendo pares ordenados, e escrevemos a notação (u, v) ao invés de $\{u, v\}$. Ao decorrer deste trabalho, iremos utilizar a denominação grafos para denotar o que definimos como grafos simples.

Um grafo G é dito *trivial* se $|V(G)| = 1$, isto é G possui um único vértice.

Um vértice u é *adjacente* a outro vértice v em G se $(u, v) \in E(G)$. Neste caso, dizemos

que u e v são *vizinhos* em G, e que a aresta $e = (u, v)$ é *incidente* a u e a v, ou que tem *extremos* u e v. Denotamos por $N(u)$ o conjunto de vértices adjacentes a u em G e tal conjunto é chamado de *vizinhança* de u, e por $N[u]$ o conjunto $N(u) \cup \{u\}$ é chamado de *vizinhança fechada* de u.

O *grau* de um vértice $v \in V(G)$, denotado por $d(v)$, é o número de arestas incidentes ao vértice v.

Um grafo H é um *subgrafo* de um grafo G se $V(H) \subseteq V(G)$ e $E(H) \subseteq E(G)$. Dado um conjunto de vértices $Y \subseteq V(G)$, $Y \neq \emptyset$, o *subgrafo induzido* por Y, denotado por $G[Y]$ é o subgrafo H de G tal que $V(H) = Y$ e $E(H)$ é o conjunto das arestas de G que têm ambos os extremos em Y. Neste trabalho assumiremos que todos os termos subgrafo que usarmos refere-se a noção de subgrafo induzido.

Um *caminho* num grafo G é um passeio $P = v_1, v_2, \ldots, v_k$ onde os $v_i's$ são vértices (dois a dois distintos), e $(v_i, v_{i+1}) \in E(G)$, $1 \leq i \leq k-1$.

Uma *corda* em P é uma aresta que liga dois vértices não-consecutivos de P. Um *caminho induzido* é um caminho sem cordas, e denotado por P_k o caminho induzido por k vértices. Dizemos que um grafo é P_k*-free* quando não contém um P_k como subgrafo.

Um passeio $v_1, \ldots, v_k, v_{k+1}$, é denominado *ciclo* quando $v_1, \ldots, v_k$ for um caminho, $k \geq 3$ e $v_1 = v_{k+1}$.

Um grafo G é dito *cíclico* quando G contém um ciclo como subgrafo. Caso contrário, dizemos que G é *acíclico*.

Um conjunto S é *maximal (minimal)* em relação a uma determinada propriedade P se S satisfaz P, e todo conjunto S' que contém propriamente S (que está contido propriamente em S) não satisfaz P.

Um grafo G é *completo* se quaisquer dois vértices distintos de G são adjacentes. Denotamos por K_n o grafo completo com n vértices.

Um conjunto de vértices I de um grafo G é um *conjunto independente* se $G[I]$ é um grafo sem arestas. Um conjunto independente I é dito *maximal* se para qualquer conjunto independente I' tal que $I \subseteq I'$ então $I = I'$. Definimos por $\alpha(G)$ o tamanho do *conjunto independente máximo*, isto é:

$$\alpha(G) = max\ \{\ |V'|\ /\ V' \subseteq V\ e\ V'\ é\ um\ conjunto\ independente\ de\ G\}.$$

Um conjunto de vértices C de um grafo G é uma *clique* se $G[C]$ é um grafo completo.

Denotamos por K_k uma clique de k vértices. Uma clique C é dita *maximal* se para qualquer clique C' tal que $C \subseteq C'$ então $C = C'$. Denotamos por $\omega(G)$ o *tamanho da clique máxima*, isto é:

$$\omega(G) = max\ \{\ |V'|\ /\ V' \subseteq V\ e\ V'\ é\ uma\ clique\ de\ G\ \}$$

Um grafo é dito *bipartido* quando seu conjunto de vértices pode ser particionado em dois subconjuntos A e B, de forma que não haja arestas entre vértices de um mesmo conjunto. Um grafo é dito *bipartido completo* quando seu conjunto de vértices pode ser particionado em dois subconjuntos A e B com todas as arestas entre A e B.

Um grafo G é *conexo* se para todo par de vértices distintos v e w de G existe um caminho de v a w. Caso contrário, G é dito *desconexo*. Um *componente conexo* de G é um subgrafo maximal conexo de G.

Sejam $v, w \in V(G)$. A *distância* entre v e w em G, denotada por $d_G(v, w)$ é o comprimento do menor caminho entre v e w, em G.

Uma *árvore* T é um grafo acíclico e conexo. Uma árvore T é denominada *enraizada* quando algum vértice $v \in V(T)$ é escolhido como especial. Este vértice é então chamado de *raiz* da árvore.

Sejam v, w dois vértices de uma árvore enraizada T de raiz r. Suponhamos que v pertença ao caminho de r a w em T, então denominamos v *ancestral* de w e w *descendente* de v. Temos também que se $(v, w) \in E(T)$ então v é *pai* de w, denotada por *pai(w)*, sendo w o *filho* de v, denotada por *filho(v)*.

Dois vértices que possuem o mesmo pai são chamados de *irmãos*. A raiz de uma árvore não possui pai, enquanto que todo vértice $v \neq r$ possui um único. Uma *folha* é um vértice que não possui filhos.

Contrair uma aresta $e = (x, y)$ é fazer a remoção de e do grafo original, e juntar seus extremos, de tal maneira que o vértice resultante seja incidente as arestas originalmente incidentes a x e y.

1.2 Complexidade de Algoritmos

Um problema algorítmico π consiste de um conjunto D de todas as possíveis entradas para o problema, chamado *conjunto de instâncias*, e de uma *questão* Q sobre estas ins-

tâncias. Resolver um problema algorítmico é desenvolver um algoritmo cuja entrada é uma instância do problema e cuja saída é uma resposta à questão do problema.

Um problema é dito de *decisão* quando a questão exige uma resposta do tipo SIM ou NÃO. Como exemplo, seja π o seguinte problema: "Dado um grafo G, reconhecer se G é um cografo". O conjunto de instâncias de π é obviamente o conjunto de todos os grafos. O problema π pode ser assim esquematizado:

Instância genérica de π: um grafo G.

Questão: G é cografo?

Fica evidente que o problema π acima é um problema de decisão, em particular, um *problema de reconhecimento*. Resolver π significa elaborar um algoritmo de reconhecimento de cografos.

Dizemos que um algoritmo é *polinomial* quando sua complexidade de tempo (medida do número de passos que o algoritmo efetua) é uma função polinomial no tamanho da sua entrada. Os problemas de decisão para os quais existem algoritmos polinomiais constituem a classe P. Tais problemas são chamados *polinomiais*.

Um problema de decisão é *não-determinístico polinomial* quando qualquer instância que produz resposta SIM possui um *certificado* sucinto, isto é, uma comprovação de que a resposta SIM é de fato verificável em tempo polinomial no tamanho da instância. Esta classe de problemas de decisão é a classe *NP*.

A classe *Co-NP* é formada pelos problemas que possuem um certificado sucinto para as instâncias que produzem resposta NÃO.

Sejam $\pi_1(D_1, Q_1)$ e $\pi_2(D_2, Q_2)$ dois problemas de decisão. Uma *transformação* ou *redução polinomial* de π_1 em π_2 é uma função $f : D_1 \longrightarrow D_2$ tal que as seguintes condições são satisfeitas:

1. f pode ser calculada em tempo polinomial;
2. para toda instância $I \in D_1$, tem-se que I produz resposta SIM para π_1 se, e somente se, $f(I)$ produz resposta SIM para π_2.

Um problema de decisão π pertence à classe *NP-completo* quando as seguintes condições são satisfeitas:

1. $\pi \in NP$;

2. para todo problema $\pi' \in NP$ existe uma transformação polinomial de π' em π.

Um problema pertencente à classe NP-completo é chamado NP-completo. Para provar que um certo problema π é NP-completo, basta mostrar que $\pi \in NP$ e que existe uma transformação de um problema NP-completo π' em π.

Analogamente, prova-se que um problema de decisão π pertence à classe *Co-NP-completo* (e, neste caso, π é dito Co-NP-completo) quando $\pi \in Co - NP$ e existe um problema π' (Co-)NP-completo tal que:

1. se π' é NP-completo, existe uma função f que pode ser calculada em tempo polinomial tal que para toda instância I' de π', tem-se que I' produz SIM para π' se e somente se $I = f(I')$ produz NÃO para π;

2. se π' é Co-NP-completo, existe uma função f que pode ser calculada em tempo polinomial tal que para toda instância I' de π', tem-se que I' produz NÃO para π' se, e somente se, $I = f(I')$ produz NÃO para π.

Como fonte de referência para esta seção, indicamos [13, 29].

Capítulo 2

Decomposição em árvore

Neste capítulo apresentamos o conceito formal de decomposição em árvore, introduzido por Robertson e Seymour [23] e bastante estudado e utilizado atualmente [9, 28, 1, 6].

Uma *decomposição em árvore* de um grafo $G = \big(V, E\big)$ é um par $\big(\chi, T\big)$ onde $T = \big(I, F\big)$ é uma árvore com conjunto de vértices I e conjunto de arestas F e $\chi = \{X_i | i \in T\}$ é uma família de subconjuntos de V, um para cada vértice de T, tal que

1. $\bigcup_{i \in I} X_i = V$.
2. para todas as arestas $\big(v, w\big) \in E$ existe um $i \in I$ com $v \in X_i$ e $w \in X_i$.
3. para todo $i, j, k \in I$: se j está no caminho de i a k em T, então $X_i \cap X_k \subseteq X_j$.

   As duas primeiras restrições implicam que χ compreende uma família de subgrafos de G. Mais precisamente, cada um desses subgrafos é o subgrafo de G induzido por χ_i. A última restrição garante a estrutura em árvore da decomposição. Esta última afirmação não é tão imediata e, portanto, iremos substituí-la pela seguinte:

4. para todo $v \in V$, $T_v = \{i \in I : v \in \chi_i\}$ induz uma árvore.

Vale observar que a decomposição em árvore de um grafo G não é única. Além disso, é facil ver que todo grafo G possui uma decomposição em árvore *trivial*, que é aquela em que a árvore possui um único nó, e que cada X_i contém todos os vértices do grafo.

No decorrer do texto, usaremos a expressão DEA_G como abreviação para decomposição em árvore de um grafo G. Quando não houver ambiguidade, usaremos simplesmente DEA. Além disso, para diferenciar vértices do grafo G e vértices de T numa DEA_G, vamos chamar os vértices i de T de nós e seus correspondentes X_i de bolsas.

Seja G um grafo. A *largura* de uma DEA_G $D = \big(\chi, T\big)$ é definida como $LA(D) = max_{i \in I}|X_i| - 1$. É fácil verificar que qualquer grafo G possui uma decomposição em árvore trivial, cuja largura é igual ao número de vértices menos um. No entanto, tal decomposição não nos fornece muita informação sobre a similaridade entre G e uma árvore, tornando esta de pouco interesse. Portanto, uma questão interessante é encontrar uma decomposição em árvore de um grafo G tal que sua largura seja a menor possível dentre todas as decomposições em árvore de G. Tal medida é denominada largura em árvore de G.

Assim, a *largura em árvore de um grafo* $G = \big(V, E\big)$ é definida como a menor largura de uma DEA_G isto é, $LA\big(G\big) = min\big(LA(D)\big)$.

Em outras palavras, a largura (ou treewidth) de uma DEA_G é o tamanho máximo de uma bolsa nesta DEA_G menos um. Enquanto que a largura em árvore do grafo é a menor largura dentre todas as possíveis decomposições em árvore de G.

A largura em árvore de um grafo G procura estabelecer uma medida de similaridade entre uma árvore e um grafo: quanto menor a largura em árvore de um grafo, mais semelhante o grafo é de uma árvore; quanto maior a largura em árvore de um grafo, mais conexo o grafo será. Quando a largura de uma DEA for igual à largura em árvore do grafo, isto é, $LA\big(D\big) = LA\big(G\big)$, então essa é dita *mínima* ou *ótima*. Neste caso, também diremos que esta DEA *realiza* a largura em árvore de G.

Na Figura 2.1 veja exemplos de decomposições em árvore e suas respectivas larguras.

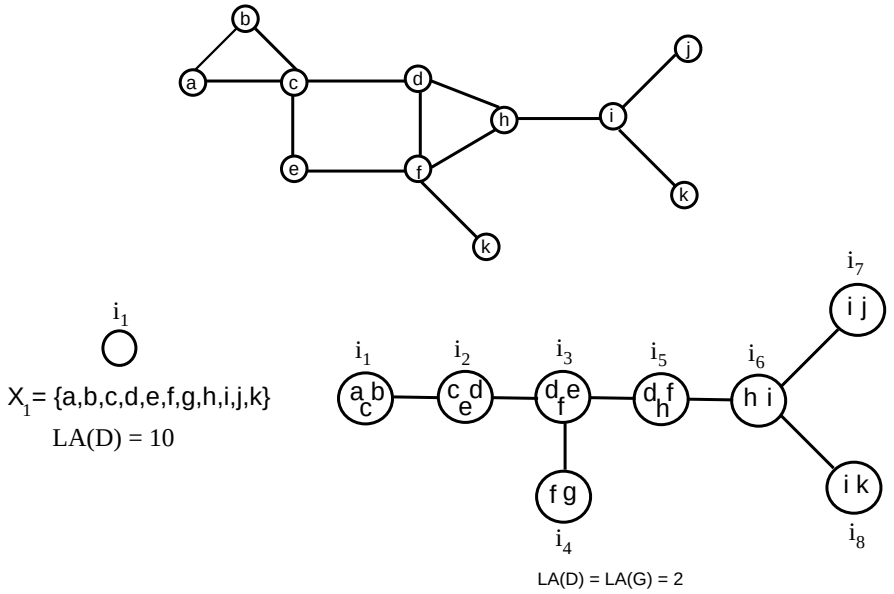

Figura 2.1: Exemplos de DEA

2.0.1 O problema de decomposição em árvore

Uma questão interessante que surge é saber a largura em árvore de um grafo G ou, em outras palavras, temos o seguinte problema de decisão:

Problema: Largura em árvore:
Entrada: Um grafo G e um inteiro positivo k
Questão: G possui largura em árvore no máximo k?

Foi demonstrado em [6] que tal problema é NP-completo.

Para o caso em que k é constante, o problema de determinar se G possui largura no máximo k pode ser decidido em tempo linear [6, 21] e, no caso afirmativo, encontra uma decomposição em árvore de G com largura no máximo k.

Tais resultados podem ser encontrados no Apêndice deste trabalho.

Na verdade, muitas das classes de grafos que possuem solução em tempo polinomial para problemas que são NP-completos para grafos gerais estão contidas na classes de grafos com largura em árvore limitada por alguma constante inteira k. Algumas das classes com largura em árvore limitada podem ser vistas na Tabela 2.1 a seguir.

Tabela 2.1: Algumas classes de grafos com largura em árvore limitada

Classes de grafos	Limite superior para largura em árvore máxima	Referências
Árvores, florestas	1	[5]
Grafos série-paralelos	2	[31]
Grafos cordais com clique máxima k	$k-1$	[24, 14]
Grafos de intervalo com clique máxima k	$k-1$	[24]
Grafos arco circular com clique máxima k	$k-1$	[5]

2.1 Decomposição em árvore enraizada

Uma *decomposição em árvore enraizada* é uma DEA $D = (\chi, T)$ em que T é uma árvore enraizada. Seja $D = (\chi, T)$ uma DEA enraizada para um grafo G. Para cada nó i de T, seja T_i a subárvore de T, com raiz i. Defina $V_i = \bigcup_{f \in T_i} X_f$ e $G_i = G[V_i]$ (então se r é a raiz de T, $G_r = G$). G_i é chamado de subgrafo de G com raiz i.

É possível obter uma DEA enraizada $D_i = (\chi_i, T_i)$ para G_i (chamada decomposição em árvore enraizada) a partir de D da seguinte maneira: Seja $D = (\chi, T)$ um decomposição em árvore enraizada para um grafo G. Seja i um nó de T. Seja $D_i = (\chi_i, T_i)$ onde T_i é a subárvore de T com raiz i, e $\chi_i = \{X_f | f \in T_i\}$. Nós chamamos D_i a DEA enraizada para o subgrafo G_i enraizada no nó i.

O próximo Lema mostra que para cada nó i, D_i é uma decomposição em árvore para G_i.

Lema 2.1 *[18] Para cada nó i, D_i é uma DEA para G_i.*

Prova. Claramente, todo vértice de G_i está em pelo menos um subconjunto de X_i. Para verificar a segunda condição para decomposições em árvore, seja (v, w) uma aresta em G_i. Existe um subconjunto X_k em χ que contém ambos os extremos da aresta. Assuma que $X_k \notin S_i$, isto é, k não é descendente de i em T. Mas v também está contido em um subconjunto X_p, sendo p um descendente de i. Mas então, pela terceira condição de decomposições em árvore, v deve estar contido em X_i e o mesmo ocorre para w. Então a segunda condição é satisfeita. Finalmente, se T' é subárvore de T, então a restrição de T' para os vértices de T_i é uma subárvore de T_i então a terceira condição também está satisfeita. ■

A seguir, apresentamos algumas propriedades das decomposições em árvore de um grafo.

2.2 Algumas propriedades de DEA

As duas propriedades a seguir fornecem um limitante superior para a largura em árvore de um grafo a partir de seus subgrafos ou de suas componentes conexas.

Propriedade 2.1 *Seja G um grafo e H um subgrafo de G. Então $LA\big(H\big) \leq LA\big(G\big)$.*

Prova. Seja $D = \big(\chi, T = \big(I, F\big)\big)$, uma DEA que realiza a largura em árvore de G. Considere agora, a família $W = \big\{\bigcup_{i \in X_i} X_i | v \in X_i \cap H\big\}$. Portanto W corresponde às bolsas da DEA D que possuem os vértices pertencentes ao subgrafo H. Claramente, como χ cobre $V(G)$, W também cobre $V(H)$ e, logo, como D é uma DEA de G, $D' = \big(W, T\big)$ é uma DEA de H $\big($ D' é também chamada de subdecomposição em árvore de G limitada a H $\big)$. Como para todo $i \in I$, $W_i \in X_i$, $LA(D') \leq LA(D)$ e, consequentemente pela minimalidade de D, $LA(H) \leq LA(G)$. ■

Na Figura 2.2 é mostrado um exemplo de uma subdecomposição como descrito na prova da Propriedade 2.1.

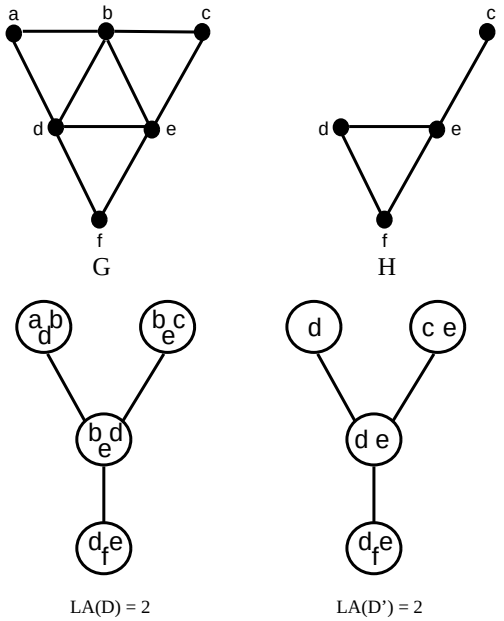

Figura 2.2: Grafo, Subgrafo, DEA e subDEA

Propriedade 2.2 *A largura em árvore de um grafo G é a máxima largura em árvore das componentes de G.*

Prova. A prova é feita de maneira construtiva. Sejam $C_1, \ldots, C_p$ p as componentes de G, e sejam $D_1, \ldots, D_p$ as decomposições em árvore que realizam a largura de $C_1, \ldots, C_p$, respectivamente. Para cada decomposição $D_j = \{\chi_j, T_j = \big(I_j, F_j\big)\}$, selecione aleatoriamente um nó $i_j \in I_j$ e considere a decomposição em árvore obtida pela união das bolsas em cada χ_j e pela adição de uma parte $X^* = \emptyset$ com um nó correspondente i^* adjacente a cada i_j, ou seja, $D^* = \big(\chi^*, T^* = \big(I^*, F^*\big)\big)$, onde

$$\chi^* = \big(\bigcup_{j=1}^{p} \chi_j\big) \cup \{X^*\},$$

$$I^* = \big(\bigcup_{j=1}^{p} I_j\big) \cup \{i^*\},$$

$$F^* = \big(\bigcup_{j=1}^{p} F_j\big) \cup \{(i^*, i_j)\text{: } i_j \text{ algum vértice em } I_j, 1 \leq j \leq p\}.$$

A Figura 2.3 ilustra tal construção. A largura de D^* pode ser então facilmente encontrada:

$$LA(D^*) = \max_{i \in I^*}\{|X_i| - 1\} = \max_{1 \leq j \leq p}\{|X_i| - 1\}\} = \max_{1 \leq j \leq p}\{LA(D_j)\}.$$

Logo, pela minimalidade de cada D_j, $LA(G) = \max_{1 \leq j \leq p}\{LA(C_j)\}$. O Lema segue. ■

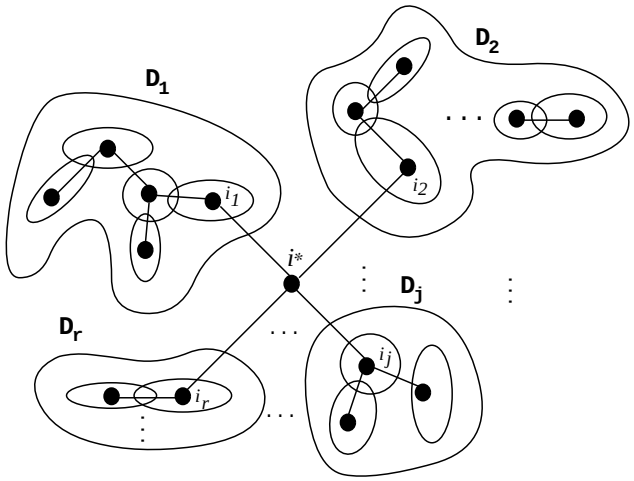

Figura 2.3: Decomposição em árvore de um grafo usando suas componentes

Um grafo H é um *menor* de um grafo G, se H foi obtido de G por uma série de contrações ou retiradas de arestas, ou eliminações de vértices. A propriedade a seguir relaciona as larguras das decomposições de um grafo G e de seu menor H.

Propriedade 2.3 *Seja G um grafo e H um menor de G. Então $LA(H) \leq LA(G)$.*

Prova. Se H foi obtido apenas por retiradas de arestas ou eliminações de vértices, então, H é subgrafo de G, e pela Propriedade 2.1, o resultado segue.

Basta, então, mostrar que a largura em árvore de um grafo não aumenta com a contração de suas arestas. Seja $D = (\chi, T)$ uma decomposição em árvore de G. Considere então $e = (u, v)$ uma aresta de G a ser contraída para a obtenção de H, e seja w o novo vértice adicionado a G', o grafo obtido pela contração de e. Uma nova decomposição em árvore $D' = (\chi', T)$ pode então ser definida como a seguir:

$$\chi' = \{X - \{u, v\} \cup w : X \in \chi,\ u \in X \text{ ou } v \in X\} \cup \{X : X \in \chi, u \notin X \text{ e } v \notin X\}.$$

Claramente, como H tem menos vértices que G, $LA(D') \leq LA(D)$. ■

Uma família $\{T_i\}_{i \in I}$ de subconjuntos de um conjunto I satisfaz a *Propriedade de Helly* se $J \subseteq I$ e $T_i \cap T_j \neq \emptyset$ para todo $i, j \in J$ implica que $\bigcap_{j \in J} T_j \neq \emptyset$.

Na Figura 2.4 são dadas subárvores que satisfazem a Propriedade Helly para uma dada árvore T.

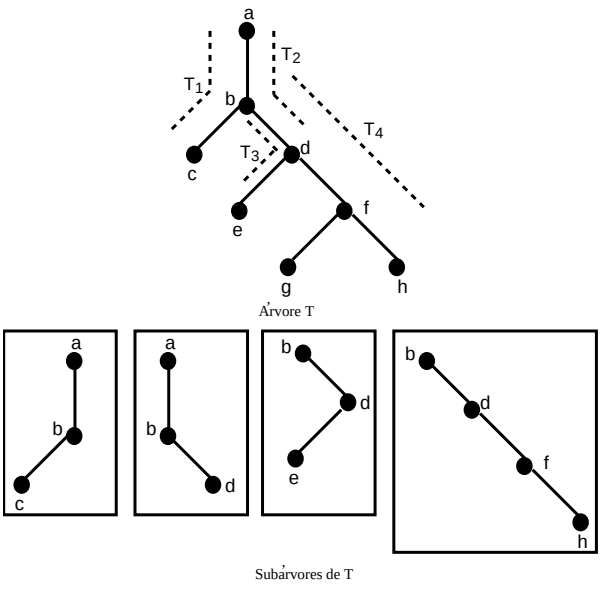

Figura 2.4: Subárvores que satisfazem a Propriedade de Helly

Dada uma árvore T e uma subárvore T_i de T, então temos o seguinte resultado.

Propriedade 2.4 *[16] Uma família $\{T_i\}_{i \in I}$ de subárvores de T satisfaz a Propriedade de Helly.*

Prova. Suponha que $T_i \cap T_j \neq \emptyset$ para todo $i, j \in T$. Para $\left|T\right| \leq 2$ a prova é trivial. Considere, então, três vértices a, b e c em T. Seja S o conjunto de índices s tal que T_s contém pelos menos dois vértices. Denote por P_{ab} , P_{bc} e P_{ac} os caminhos simples em T conectando a com b, b com c e a com c, respectivamente. Note que sendo T uma árvore, T não possui ciclos, então temos a interseção em vértices $P_{ab} \cap P_{bc} \cap P_{ac} \neq \emptyset$. Além disso, cada T_s, $s \in S$, contém pelo menos um desses caminhos. Portanto, $\bigcap_{s \in S} T_s \supseteq P_{ab} \cap P_{bc} \cap P_{ac} \neq \emptyset$.

A prova do Teorema é feita por indução no tamanho de J. Assumimos que $\left[T_i \cap T_j \neq \emptyset, \forall i, j \in J\right] \rightarrow \bigcap_{j \in J} T_j \neq \emptyset$, para todo conjunto de índices de J de tamanho $\leq k$. Note que isso é certamente verdade para $k = 2$, $\left(\left[T_1 \cap T_2 \neq \emptyset\right] \rightarrow \bigcap_{i \in J} T_j = T_1 \cap T_2 \neq \emptyset\right)$. Considere a família de subárvores $J' = \left\{T_1, ..., T_{k+1}\right\}$. Pela hipótese de indução, existem três vértices a, b e c em T tal que: $a \in \bigcap_{j=1}^{k} T_j$, $b \in \bigcap_{j=2}^{k+1} T_j$ e $c \in T_1 \cap T_{k+1}$. Usando o fato mencionado anteriormente no primeiro parágrafo da prova e observando que cada T_j contém pelo menos dois dos vértices a, b e c, podemos concluir que $\bigcap_{j=1}^{k+1} T_j \supseteq P_{ab} \cap P_{bc} \cap P_{ac} \neq \emptyset \rightarrow \bigcap_{j=1}^{k+1} T_j \neq \emptyset$.

■

2.2.1 DEA para algumas classes especiais de grafos

Como foi mencionado inicialmente, a DEA de um grafo representa, de certo modo, o quanto um grafo se assemelha de uma árvore. O parâmetro utilizado para medir tal semelhança é a largura em árvore do grafo. A seguir, apresentamos algumas classes de grafos para os quais se conhece a largura em árvore.

Parece natural intuir que a largura em árvore de uma floresta é muito pequena, uma vez que cada componente de G é, por si só, uma árvore.

Teorema 2.1 *Um grafo G tem largura no máximo 1 se, e somente se, G é uma floresta.*

Prova. ($\Rightarrow$) Assuma que G tem largura em árvore um. Então G possui um vértice de grau máximo um. Removendo-se este vértice de G, a largura em árvore de G não aumenta. Portanto, podemos remover recursivamente vértices de grau no máximo um de G (sem aumentar a largura), até remover todos os vértices de G. Implicando em G ser uma floresta.

($\Leftarrow$) Considere uma árvore arbitrária G. Construímos uma decomposição em árvore para G com largura em árvore um como segue:

1. Transforme G em uma árvore enraizada, com raiz r e oriente suas arestas a partir de r.
2. Para cada aresta orientada (u, v) de G, crie um nó rotulado $\{u, v\}$ em T.
3. Crie uma aresta ligando os nós rotulados $\{u_1, v_1\}$ e $\{u_2, v_2\}$ se, e somente se, $v_1 = u_2$.

Pode-se verificar que T corresponde a uma decomposição em árvore de G de largura um. Observe que se G for uma floresta, as componentes conexas de G são árvores e, portanto, pela Propriedade 2.2 G possui largura em árvore no máximo um.

■

O Teorema anterior nos fornece um limite inferior para a largura em árvore de grafos contendo ciclo. Assim, qualquer grafo que possua ao menos um ciclo, possui largura em árvore pelo menos dois. O Lema a seguir, mostra que todo ciclo tem largura em árvore dois.

Lema 2.2 *Todo ciclo tem largura em árvore* 2.

Prova. Seja C um ciclo, $C = \{v_0, e_1, v_1, e_2, v_2, ..., e_{l-1}, v_{l-1}, e_l, v_0\}$. Para provar que $LA(C) \leq 2$, definimos uma DEA $D = \big(\chi, T = (I, F)\big)$, onde

$$I = \{1, 2, ..., l\},\ X_k = \{\{u, v : (u, v) = e_k\} \cup v_0\}_{1 \leq k \leq l} \text{ e } F = \{(i, i+1) : 1 \leq i \leq l-1\}.$$

Então (χ, T) é uma decomposição em árvore de C com largura dois. Pelo Teorema 2.1, C tem largura maior do que um e o resultado segue. ■

Um grafo $G = (V, E)$ é dito *bipartido completo especial* quando seu conjunto de vértices V puder ser particionado em dois subconjuntos A e B e $\forall v \in A$ e $\forall w \in B$, $(v, w) \in E$. Vale observar que nesta definição de bipartido completo não se faz restrição quanto a não existência de arestas entre vértices de um mesmo conjunto, isto é, A e B não são obrigatoriamente conjuntos independentes.

O Lema a seguir mostra que toda clique de um grafo G deve estar contida em alguma bolsa de qualquer decomposição em árvore de G.

Lema 2.3 *[22] Seja $G = (V, E)$ um grafo e $T = (I, F)$ uma DEA.*

1. **Lema de continência da clique**: *Seja $C \subseteq V$ uma clique em G. Então $\exists i \in I$ tal que $C \subseteq X_i$.*

2. **Lema de continência do subgrafo bipartido completo especial**: *Sejam A e B subgrafos bipartidos completos especiais de G, isto é, cada vértice em A é adjacente a cada vértice em B. Então, $\exists i \in I : A \subseteq X_i$ ou $B \subseteq X_i$.*

Prova.

1. Seja C uma clique em G. Para todo par de vértices x e y de C existe um subconjunto (ou bolsa) X_j contendo ambos x e y. Além disso, para todo vértice $v \in C$ o conjunto de bolsas X_i contendo v forma uma subárvore de T, denotada por T_v. A família F de subárvores T_v, $v \in C$ é uma família de subávores de T. Pela Propriedade 2.4 sabemos que uma família de subárvores de uma árvore T satisfaz a Propriedade de Helly. Assim, existe $i \in I$ tal que $\forall v \in C$, $i \in T_v$ $\big(i$ é o vértice de interseção de todas as subárvores da família $F\big)$. Portanto, podemos concluir que $\exists i \in I$ tal que $C \subseteq X_i$.

2. Seja a DEA $\big(\{X_i \in I\}, T = (I, F)\big)$ do grafo $G = (V, E)$, com bipartição completa especial A, B. Suponha que para todo $i \in I : B \not\subseteq X_i$ (então temos que encontrar $A \subseteq X_i$ para algum $i \in I$). Faça $T_v = \{i \in I | v \in X_i\}$. Considere a família $\{T_v | v \in B\}$ de subárvores de T. Como $\bigcap_{v \in B} T_v = \emptyset$, segue pela Propriedade 2.4 que existem $b_1, b_2 \in B$ tal que T_{b_1} e T_{b_2} são disjuntas em vértice. Considere o único caminho de T conectando T_{b_1} e T_{b_2}, e sejam k e l os vértices dos extremos desses caminhos. (Veja Figura 2.5). Cada $a \in A$ deve estar contido em um conjunto X_i com $i \in T_{b_1}$ e em um conjunto X_i com $j \in T_{b_2}$. Então $a \in X_k$. Portanto $A \subseteq X_k$.

■

Corolário 2.1 *Seja $G = (V, E)$ um grafo. Então $LA(G) \geq \omega(G) - 1$.*

Prova. A prova é consequência imediata do Lema de continência da clique e da definição de largura de um grafo.

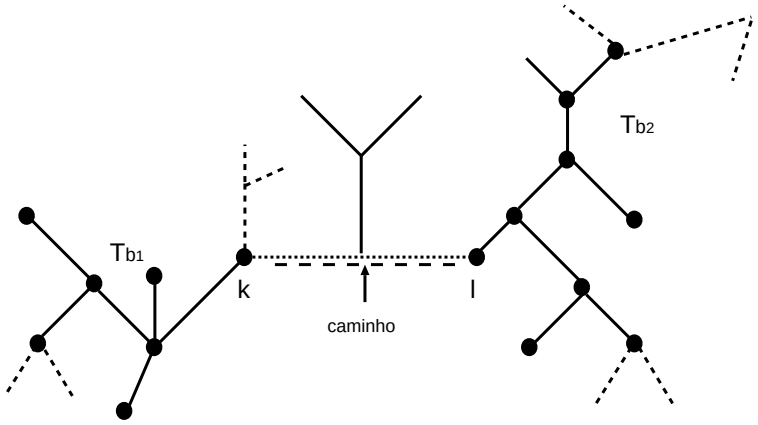

Figura 2.5: Uma ilustração da prova do Lema 2.3

$$LA(G) \geq max_{i \in I}\big(|X_i|\big) - 1$$

$$max_{i \in I}\big(|X_i|\big) \geq \omega(G)$$

$$LA(G) \geq \omega(G) - 1$$

■

Capítulo 3

Largura em árvore de classes especiais de grafos

Neste capítulo consideramos algumas classes de grafos especiais. Mais precisamente, na Seção 3.1 consideramos os grafos cordais, seguido de sua caracterização a partir de um esquema de eliminação perfeito. Na Seção 3.1.2 caracterizamos os grafos cordais em termos de sua decomposição em árvore. Além disso, na Seção 3.1.3 apresentamos um algoritmo que constrói uma decomposição em árvore de grafos cordais. Na Seção 3.2 abordamos uma subclasse dos grafos cordais, as k-árvores. Na Seção 3.3 analisamos uma heurística de triangularização muito conhecida, pois veremos que qualquer triangularização do grafo define uma decomposição em árvore de um grafo com a mesma largura em árvore. Na Seção 3.4 mostramos como transformar uma DEA qualquer em uma boa decomposição em ávore, uma DEA de grande interesse pois facilita o projeto de algoritmos.

3.1 Grafos cordais

Um grafo não direcionado $G = \big(V, E\big)$ é chamado *cordal* se cada ciclo induzido de tamanho estritamente maior que 3 possui uma corda, isto é, uma aresta unindo dois vértices não consecutivos. Como G não possui subgrafos induzidos isomorfos a C_n, $n > 3$, o maior ciclo induzido de G é um triângulo. Ser cordal é uma propriedade hereditária, inerente a todos os subgrafos induzidos de G.

Na literatura, grafos cordais podem também ser chamados de *triangularizados*, *circuito rígido* e *grafos de eliminação perfeita*.

Para o entendimento do algoritmo que encontra uma DEA para a classe de grafos cordais, são necessárias algumas definições e caracterizações que serão dadas nesta seção.

3.1.1 Algumas definições

Um vértice $x \in G$ é chamado *simplicial* se seu conjunto vizinhança $N(x)$ induz um subgrafo completo de G, isto é, $N(x)$ é uma clique (não necessariamente maximal). Dirac [10] e mais tarde Lekkerkerker and Boland [20] provaram que um grafo cordal sempre tem um vértice simplicial (na verdade, pelo menos dois).

Seja $\sigma = [v_1, ..., v_n]$ uma ordenação dos vértices e denote G_i o subgrafo restante depois da eliminação de $v_1, ..., v_{i-1}$ de G $\Big(G_i = G\big[V - v_1, ..., v_{i-1}\big]\Big)$. Definimos σ como sendo um *esquema de eliminação perfeito* se v_i é um vértice simplicial no grafo G_i, $\forall 1 \leq i \leq n$.

Sejam a e b dois vértices não adjacentes de um grafo G. Um subconjunto $S \subset V$ é um *separador de vértices* a e b (ou *ab-separador*) se a remoção de S do grafo $\Big(G' = V - S\Big)$ separa a e b em componentes distintas. Se nenhum subgrafo de S é um *ab*-separador, então S é um separador minimal para a e b.

A Figura 3.1 demonstra alguns exemplos das definições dadas até agora para um grafo cordal.

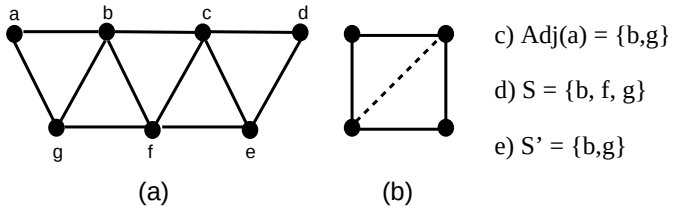

Figura 3.1: (a)Grafo cordal; (b) Grafo não-cordal; (c) Conjunto adjacência do vértice simplicial a; (d) S é um *ac*-Separador; (e) S' é um *ac*-Separador Minimal

Os separadores minimais de um grafo também caracterizam um grafo cordal, conforme o Lema 3.1 a seguir.

Lema 3.1 *[10] G é um grafo cordal se, e somente se, todo separador minimal induz um subgrafo completo em G.*

Prova. Seja S um *ab*-separador minimal e $G[A]$ e $G[B]$ as componentes conexas de $G[V - S]$ contendo a e b, respectivamente. A Figura 3.2(a) representa G através do separador S e as duas componentes A e B de $G - S$. Sendo S um separador minimal, então todo $x \in X$ é adjacente a algum vértice em A e a algum vértice em B. Caso contrário, esse vértice pode ser eliminado de S e um novo *ab*-separador S' menor que S pode ser construído, tal que $S' = S - \{x\}$. Portanto, para qualquer $x, y \in S$, existem

os caminhos $P_1 = [x, a_1, ..., a_r, y]$ e $P_2 = [y, b_1, ..., b_t, x]$ onde cada $a_i \in A$ e $b_j \in B$, para $1 \leq i \leq r$ e $1 \leq j \leq t$. Esses caminhos são escolhidos para terem o menor comprimento possível. Verifique um esboço de P_1 e P_2 do grafo G na Figura 3.2(b). Segue que $P_1 \cup P_2 = [x, a_1, ..., a_r, y, b_1, ..., b_t, x]$ é um ciclo em G cujo comprimento é estritamente maior que 3, implicando que deve existir uma corda. Mas $a_i b_j \notin E$ pela definição de Separador; $a_i a_j \notin E$ e $b_i b_j \notin E$ pela minimalidade de r e t. Como exemplo, veja a Figura 3.2(c). Suponha que a corda seja (a_1, a_3) então poderíamos eliminar as arestas (a_1, a_2) e (a_2, a_3) do caminho P_1. Se a corda fosse (b_1, b_3), poderíamos eliminar as arestas (b_1, b_2) e (b_2, b_3). Assim, a única corda possível é $xy \in E$. Como isso é válido para qualquer $x, y \in S$, segue que S induz um subgrafo completo em G.

Seja $C = [a, x, b, y_1, y_2, ..., y_k, a]$ um ciclo do grafo G, $k \geqslant 1$. Como qualquer ab-separador minimal deve conter os vértices x e y_i, para $1 \leq i \leq k$, e sabendo que qualquer separador de um grafo cordal induz um subgrafo completo, $(x, y_i) \in E$ e $(y_i, y_i) \in E$, para $1 \leq i, j \leq k$, logo o ciclo sempre terá uma corda. Veja na Figura 3.3 o ciclo C de um grafo G. ■

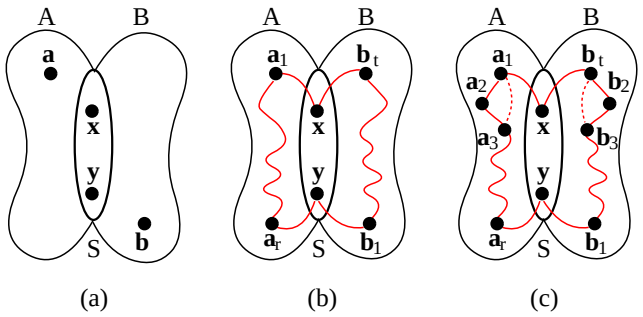

Figura 3.2: (a) O grafo G separado em duas componentes A e B pelo separador S; (b) Os caminhos P_1 e P_2 do grafo G; (c) As cordas proibidas pela minimalidade de r e t

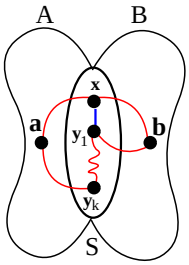

Figura 3.3: Ciclo C da prova do Lema 3.1

Podemos observar que isso também segue para $r = t = 1$, implicando que para todo $x, y \in S$, existem vértices em A e B que são adjacentes a ambos x e y. Um resultado mais

forte é dado no Lema 3.2.

Lema 3.2 *[16] Para qualquer separador minimal S de um grafo cordal $G = (V, E)$ existe um vértice v em cada componente conexo de $G[V - S]$ tal que $S \subseteq N(v)$.*

Prova. Seja $G = (V, E)$ um grafo cordal e S um separador minimal de G. A prova é feita por indução no tamanho de S. Se $S = x$ então o lema é trivial (Figura 3.4(a)). Seja então $S = x_1, ..., x_k$ suponha que existe um vértice v tal que $S - x_k \subseteq N(v)$ (Figura 3.4(b)). Se $x_k \in N(v)$ a prova está completa, pois v é adjacente a todos os vértices de S. Suponha então ao contrário, que $x_k \notin N(v)$. Logo, deve existir um outro vértice u pertencente à mesma componente de v e adjacente a x_k. Do contrário, S não seria minimal, isto é, x_k poderia ser eliminado de S. Assuma ainda, sem perda de generalidade, que u é adjacente a v (Figura 3.4(c)). Pelo Lema 3.1, S induz um subgrafo completo, então x_k é adjacente a todos os vértices de S. Para cada x_i, $1 \leq i \leq k - 1$, $C_i = (x_i, v, u, x_k, x_i)$ induz um ciclo em G, e logo como G é cordal e $(v, x_k) \notin E$, deve existir uma corda unindo u e x_i. Observe um ciclo C_i para i qualquer obedecendo a $1 \leq i \leq k$ na Figura 3.4(d). Como isso é válido para qualquer $x_i \in S$, então u é adjacente a todos os vértices em S. A prova está concluída. ■

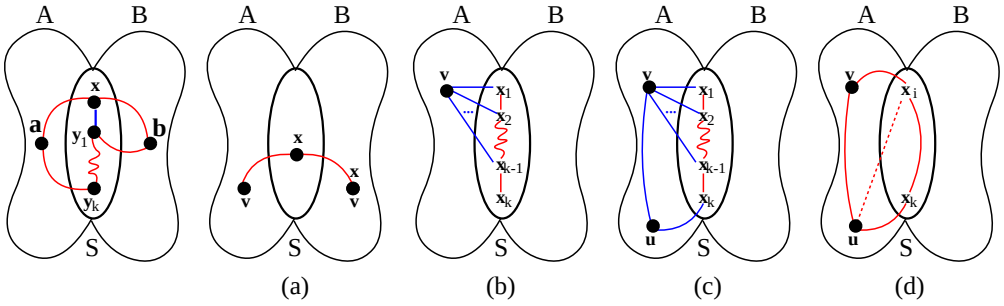

Figura 3.4: Ciclo C da prova do Lema 3.2

Lema 3.3 *[10] Todo grafo cordal $G = (V, E)$ tem um vértice simplicial. Além disso, se G não é uma clique, então ele tem dois vértices simpliciais não adjacentes.*

Prova. O Lema é trivial se G é completo. Assuma que G tem dois vértices não adjacentes a e b e que o Lema é verdade para todos os grafos com menos vértices que G. Sejam S um separador de vértices minimal para a e b com $G[A]$ e $G[B]$ as componentes de $G[V - S]$ contendo a e b, respectivamente. Por indução, o subgrafo $G[A + S]$ tem dois vértices

simpliciais não adjacentes. Um desses tem que estar em A (pois S induz um subgrafo completo); ou $G\big[A+S\big]$ é ele mesmo completo e qualquer vértice de A simplicial em $G\big[A+S\big]$. Além disso, considerando $N\big(A\big)$, o conjunto adjacência dos vértices em A, $N\big(A\big) \subseteq A+S$ então um vértice simplicial x em $G\big[A+S\big]$ é simplicial em todo G. Similarmente, B contém um vértice y simplicial de G. Assim, x e y, são dois vértices não simpliciais e não adjacentes em G. ■

Lema 3.4 *[12] G é um grafo cordal se, e somente se, G admite um esquema de eliminação perfeito . Além disso, qualquer vértice simplicial pode iniciar um esquema perfeito.*

Prova. ($\Rightarrow$) De acordo com o Lema 3.3 se G é cordal, então ele tem um vértice simplicial, dito x. Sendo $G\big[V-x\big]$ também cordal e menor que G, ele tem, por indução, um esquema perfeito que, quando unido ao sufixo x, forma um esquema perfeito para G.

($\Leftarrow$) Sejam C um ciclo de G e x um vértice de C de menor índice em um esquema perfeito. Como os vizinhos de x têm que formar uma clique devido à minimalidade de x e $\big|N(x) \cap C\big| \geqslant 2$, podemos garantir uma corda em C. ■

Uma aplicação do esquema de eliminação perfeito, além do reconhecimento dos grafos cordais, é que este pode ser utilizado para encontrar cliques maximais de um grafo cordal em tempo polinomial, enquanto que o mesmo problema é NP-completo para grafos gerais. Uma observação importante é que os grafos cordais podem ter apenas um número linear de cliques maximais, enquanto que um grafo que não seja cordal pode ter um número exponencial. Para listar todas as cliques maximais de um grafo cordal, basta encontrar um esquema de eliminação perfeito, formar uma clique para cada vértice v junto com os seus vizinhos que ocorrerem depois de v na ordenação, e testar se cada um dos resultados obtidos é uma clique maximal. Como temos apenas n vértices no esquema, teremos construído no máximo n cliques maximais. A maior clique maximal será, portanto, a clique máxima do grafo, e, já que grafos cordais são perfeitos, o tamanho de sua maior clique é igual ao número cromático do grafo cordal. Assim, os problemas de clique máxima e de coloração são resolvidos polinomialmente para grafos cordais.

Portanto, quando um grafo é cordal, a existência do esquema de eliminação perfeito permite resolver alguns problemas que são NP-completos em grafos gerais. A seguir, caracterizaremos os grafos cordais em termos de sua decomposição em árvore pois, como foi dito inicialmente neste trabalho, muitos problemas NP-completos podem ser resolvidos

polinomialmente para certas classes de grafos considerando algumas características de sua decomposição em árvore.

3.1.2 Decomposição em árvore da classe dos grafos cordais

O Teorema a seguir caracteriza os grafos cordais em termos de sua DEA.

Teorema 3.1 *[18] Um grafo $G = \big(V, E\big)$ é cordal se, e somente se, admite uma DEA $D = \big(\chi, T = \big(I, F\big)\big)$ onde χ corresponde à família de cliques maximais de G.*

Prova. ($\Rightarrow$) Essa prova é construtiva e mostra como construir uma decomposição em árvore onde os vértices correspondem às cliques maximais. Suponha que $G = \big(V, E\big)$ seja cordal. A prova da condição necessária é feita por indução no tamanho de G. Assuma, portanto, que a condição é válida para todos os grafos com menos vértices que G.

Como base da indução, se G é completo, então T é composta de somente um nó T_1 com $X_1 = V$, tal que $X_1 \in \chi$ e $T_1 \in I$.

Se G é desconexo com componentes $C_1, C_2, ..., C_r$ então existe uma DEA $D_j = \big(\chi_j, T_j = (I_j, F_j)\big)$ para cada componente C_j, onde $1 \leq j \leq r$, tal que χ_j corresponde à família de cliques maximais, pela hipótese de indução. Para cada decomposição D_j, selecione aleatoriamente um nó $i_j \in I_j$. Considere agora a decomposição em árvore obtida pela união das (sub)árvores T_i e pela adição de um nó i^*, tal que X_{i^*} seja igual a clique maximal de alguma das componentes (Figura 3.5). Suponha, sem perda de generalidade, que a clique maximal W escolhida pertença à componente C'. A nova árvore que então se forma respeita as restrições de decomposição em árvore, já que $W \cap C_j = \emptyset$ para todo $C_j \neq C'$. Além disso vale observar que no caso em que $C' = C_j$, W corresponde à uma das partes na decomposição.

Assuma agora que G é conexo e não completo, e seja w um vértice simplicial de G. Defina agora $W = w \cup N(w)$. W é claramente uma clique maximal de G. Sejam ainda $U = u \in W : N(u) \subset W$ e $Y = W - U$ (isto é, vértices que estão em W e que possuem ao menos um vizinho fora de W). Considere agora o grafo $G' = G\big[V - U\big]$, que é também cordal, pois ser cordal é uma propriedade hereditária. G' tem menos vértices que G, portanto pela hipótese de indução admite uma DEA $D' = \big(\chi', T' = (I', F')\big)$ satisfazendo o teorema, ou seja, χ' corresponde a uma família de cliques maximais.

Seja então K a clique maximal de G' contendo Y. Para a construção de uma decomposição $D = \big(\chi, T = (I, F)\big)$ de G a partir de D' e G', devemos considerar dois

casos:

1. $Y = K$: Nesse caso, seja $i \in I'$ tal que $X_i = \{K\}$. Logo, basta substituir K por W em X_i na DEA D. A DEA D é a mesma DEA D' exceto pelo fato que $X_i = \{W\}$.

2. $Y \subset K$: Como no caso anterior, seja novamente $i \in I'$ tal que $X_i = K$. A nova decomposição é obtida acrescentado um novo nó i' adjacente a i tal que $X_{i'} = W$.

($\Leftarrow$) Seja $G = \big(V, E\big)$ um grafo com uma DEA $D = \big(\chi, T = \big(I, F\big)\big)$ tal que χ corresponda à família de cliques maximais de G. Inicialmente, vale observar que se $\big(u, v\big) \in E$ então as subárvores T_u e T_v se interceptam, já que a clique maximal que contém u e v deve estar em T_u e T_v, pela propriedade de DEA.

Suponha agora, por absurdo, que G contenha um ciclo sem cordas $C = \{v_0, v_1, ..., v_{k-1}, v_0\}$ com $k > 3$ e com subárvores correspondentes T_0, T_1, ..., T_{k-1}, isto é, subárvores que possuam algum nó contendo $v_0, v_1, ..., v_{k-1}$, respectivamente, em T. Seja $F = \{P_i\}_{0 \leq i \leq k-1}$ uma família de caminhos em T tal que $P_i \subseteq T_i$, $P_i \cap P_{(i-1) \ mod \ k} \neq \emptyset$ e $P_i \cap P_{(i+1) \ mod \ k} \neq \emptyset$. Como $T_i \cap T_{(i-1) \ mod \ k} \neq \emptyset$ e $T_i \cap T_{(i+1) \ mod \ k} \neq \emptyset$, pelas arestas em C, tais caminhos sempre existem. Vale notar que $P_{(i-1) \ mod \ k} \cap P_i \cap P_{(i+1) \ mod \ k} = \emptyset$, já que, caso contrário, a interseção entre os caminhos implicaria numa clique contendo $v_{(i-1) \ mod \ k}$, v_i e $v_{(i+1) \ mod \ k}$, o que representa uma corda em C, violando a hipótese de absurdo. Além disso, pela Propriedade 2.4, F deve satisfazer a Propriedade de Helly, e logo, para todo i, $T_{(i-1) \ mod \ k} \cap T_{(i+1) \ mod \ k} \neq \emptyset$, formando um ciclo (não induzido) em T. Um absurdo, já que T é uma árvore.

■

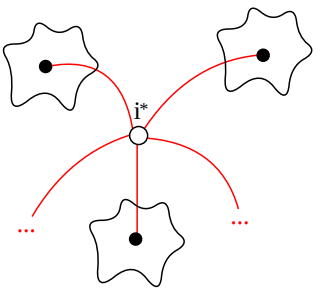

Figura 3.5: Decomposições $D'_j s$ das componentes de G unidas pelo vértice i^*

Em qualquer DEA $\big(\chi, T\big)$ e toda clique máxima Q, existe um nó $i \in I$ com $Q \subseteq X_i$. A DEA tem largura mínima. Além do mais, segue que a largura em árvore de um grafo

cordal é igual à cardinalidade da clique máxima menos um, como provaremos no Corolário 3.1 logo a seguir.

Corolário 3.1 *Se G é um grafo cordal, então $LA(G) = \omega(G) - 1$.*

Prova.

O corolário segue diretamente do Teorema 3.1, ou seja, as partes de χ da DEA mínima de G conterá somente cliques. Pelas definições dadas de largura, a largura em árvore de uma decomposição é dada por suas partes, e logo, a largura em árvore de um grafo cordal G é dada por aquela parte que contiver a maior clique, isto é, $LA(G) = \omega(G) - 1$. ■

Pelo Corolário anterior, podemos concluir que determinar a largura em árvore de um grafo cordal é uma tarefa que pode ser feita em tempo linear, bastando, para isso, determinar o tamanho da maior clique do grafo. Pode-se também inferir que grafos cordais não possuem largura limitada pois depende do tamanho de sua maior clique.

Uma *triangularização* (também chamada triangulação) ou *cordalização* é a transformação de um grafo $G = (V, E)$ em um grafo $H = (V, F)$ de modo que H seja cordal e $E \subseteq F$. O subconjunto $E_t = F - E$ é também chamado de *arestas de preenchimento* ou *fill-in* de G. Portanto, *triangularizar* é inserir arestas em um grafo G de forma a transformá-lo em um grafo cordal. Uma triangularização H de G é dita *minimal* se nenhum subgrafo de H é uma triangularização de G. Observe que $LA(G) \leq LA(H)$, considerando que toda decomposição em árvore de H é também uma decomposição em árvore de G, como vimos na Propriedade 2.1, página 11. Portanto, encontrar um fill-in mínimo de um grafo (i.e., encontrar o menor número de arestas a serem adicionadas a um grafo G de forma a torná-lo cordal) é útil para obter um limite superior para a largura do grafo. No entanto, em [32] Yannakakis mostrou que encontrar o fill-in mínimo de um grafo é um problema NP-difícil.

Exemplos de vértices simpliciais, esquema de eliminação perfeito e triangularização podem ser vistos na Figura 3.6.

A seguir, um algoritmo para encontrar uma decomposição em árvore de um grafo cordal é apresentado.

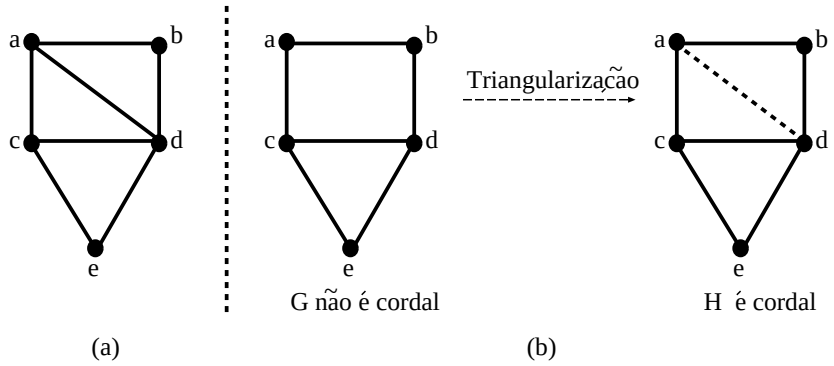

Figura 3.6: (a) Vértices simpliciais e esquema de eliminação perfeito ; (b) Triangularização

3.1.3 Decomposição em árvore de Grafos Cordais - Algoritmo

Nesta seção apresentamos um algoritmo que constrói uma decomposição em árvore mínima para um grafo cordal. É uma heurística recursiva que encontra uma decomposição em árvore utilizando separadores de vértices minimais e dividindo o grafo em subgrafos.

No que segue, se $D = \big(\chi, T\big)$ é uma DEA de um grafo G, então $T\big(D\big)$ retorna a árvore da decomposição; de modo análogo, $\chi\big(D\big)$ retorna a família de partes de D e $E\Big(T\big(D\big)\Big)$ retorna as arestas da árvore $T\big(D\big)$. A função SEP_MINIMAL(u, v, G) retorna o separador minimal dos vértices u e v. A função COMPONENTES (G) retorna uma lista contendo os conjuntos de vértices que induzem cada componente C do grafo.

Algoritmo 3.1 *[9]*

- *Algoritmo: DEA_ Cordal*
- *Entrada: Um grafo cordal* $G = \big(V, E\big)$
- *Saída: Uma decomposição em árvore* $G = \big(\chi, T\big)$ *mínima de* G

1. **Inicio**
2. **se** G *é completo* **então**
3. **retorna** $\big(\{V\}, (\{t\}, \emptyset)\big)$
4. **senão**
5. **selecione** $u, v : (u, v) \notin E,\ u, v \in V$
6. $S \leftarrow SEP_MINIMAL(u, v, G)$

7. $D \leftarrow \big(\{S\}, (\{t\}, \emptyset)\big)$

8. **para** *cada* $C \in COMPONENTES(G[V - S])$ **faça**

9. $D' \leftarrow DEA_Cordal(G[C \cup S])$

10. **selecione** t'*:* $S \subseteq X_{t'}$, $t' \in T(D')$, $X_{t'} \in \chi(D')$

11. $\chi(D) \leftarrow \chi(D) \cup \chi(D')$

12. $T(D) \leftarrow T(D) \cup T(D')$

13. $E(T(D)) \leftarrow E(T(D)) \cup \{(t, t')\}$

14. **retorna** D

15. **Fim**

Podemos fazer algumas observações sobre a estrutura do algoritmo apresentado. Pelo Lema 3.1, todo separador minimal de um grafo induz uma clique. Como cordalidade é uma propriedade hereditária, cada componente de $G - S$ é também cordal, o que justifica a recursão na linha 8.

Como a base da recursão ocorre nas linhas 2 e 3, podemos ver claramente que toda bolsa de $\chi \in D$ é uma clique. Logo, como a maior clique de G está exatamente contida em alguma bolsa $X_i \in \chi$, a DEA gerada pelo algoritmo é mínima. Resta justificar a existência de t' na linha 9: isto é imediato a partir do Lema 3.2, já que existe um vértice v em $G - S$ tal que $v \cup S$ induz uma clique. Logo, também existe um nó t' em $T_v(D')$ contendo $v \cup S$.

A Figura 3.7 exemplifica a construção de uma DEA de um grafo cordal, onde cada S_i é um separador minimal da recursividade na iteração i. Observe pelo Corolário 3.1, $LA(G) = 2$, como podemos verificar por $LA(G) = \omega(G) - 1 \Rightarrow LA(G) = 3 - 1 \Rightarrow LA(G) = 2$.

3.2 K-árvores

Uma k-árvore é um grafo que pode ser reduzido a um K_k, isto é, um grafo completo de $k + 1$ vértices, através de remoções sequenciais de vértices de grau k com vizinhos completamente conectados (uma $k + 1$-clique). A importância das k-árvores se dá pelo fato dessa classe de grafos possuir largura limitada em k. Nesta seção estudaremos algumas caracterizações básicas de k-árvores e provaremos o limite de sua largura.

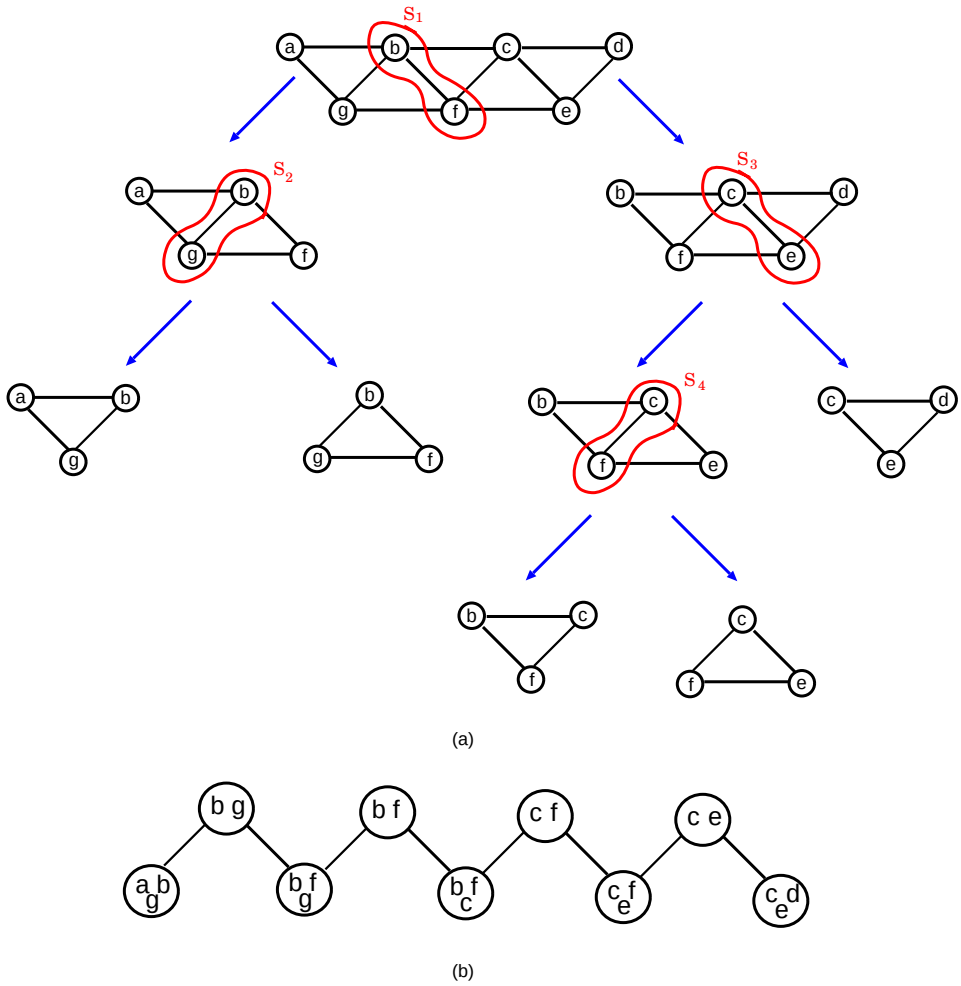

Figura 3.7: Decomposição em árvore de largura mínima de um grafo cordal

3.2.1 Caracterizações de *k*-árvores

As *k-árvores* são generalizações de árvores (que são 1-árvores). São uma classe particular de grafos cordais que podem prover uma caracterização interessante para grafos de largura limitada. O interesse em tais grafos provém da convicção de que para muitos propósitos o k mínimo para que um grafo G seja uma k-árvore parcial (um subgrafo de uma k-árvore), é uma boa medida de propriedades algorítmicas de G.

Um grafo $G = (V, E)$ pertence à classe k-árvore se ele é um grafo completo com $k+1$ vértices ou se ele tem um vértice $v \in V$ cujos vizinhos induzem uma k-clique de tamanho k, tal que $G - \{v\}$ é novamente uma k-árvore. Então, podemos dizer que uma k-árvore é definida recursivamente como a seguir: uma k-árvore com k vértices consiste em uma clique com k vértices; dada qualquer k-árvore T_n com n vértices, construímos uma k-árvore com $n+1$ vértices adicionando um novo vértice v_{n+1} a T_n fazendo-o adjacente a cada vértice de alguma k-clique de T_n e não adjacentes aos $n-k$ vértices restantes. Essa

definição recursiva define um processo de redução para k-árvores. Podemos testar se um grafo é uma k-árvore removendo um vértice v de grau k com k vizinhos completamente conectados (um k-folha). Então o grafo é uma k-árvore se, e somente se, $G - \{v\}$ for uma k-árvore (ou ainda, se após remover todos os vértices v como anteriormente, o grafo restante for uma k-clique). Observe que todas as cliques de uma k-árvore tem tamanho $k + 1$. Na Figura 3.8 apresentamos exemplos de k-árvores. Um grafo G é chamado de *k-árvore parcial* se G é um subgrafo de uma k-árvore.

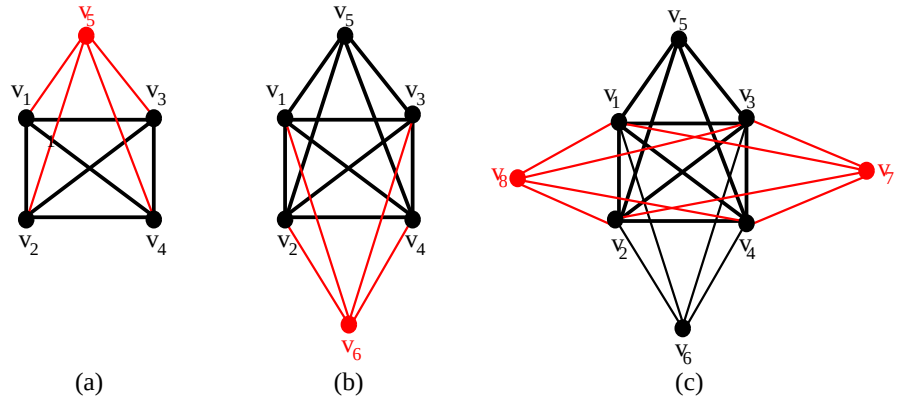

Figura 3.8: (a) 4-árvore com 4 vértices; (b) 4-árvore com 5 vértices; (c) 4-árvore com 6 vértices; (d) 4-árvore com 8 vértices

Para qualquer k-árvore parcial G, com pelo menos k vértices, existe uma k-árvore H com o mesmo número de vértices para o qual G é um subgrafo. (Neste caso dizemos que G está *imerso* em H.)

3.2.2 Lemas, Teoremas e Corolários

O Lema a seguir prova que a classe de grafos k-árvore é subclasse dos grafos cordais.

Lema 3.5 *[26] Uma k-árvore é um grafo cordal.*

Prova. Seja T_n uma k-árvore com n vértices. Pela definição de k-árvore sempre existe um vértice v_1 adjacente a uma k-clique e, logo, simplicial. Com a retirada de v_1 de T_n, temos ainda uma k-árvore T_{n-1} com $n - 1$ vértices, de onde podemos retirar um outro vértice v_2 que também é simplicial pois é adjacente a uma k-clique. Podemos perceber então que $n - k$ vértices podem ser retirados de T_n de forma iterada. Como os outros vértices fazem parte de uma k-clique eles são também simpliciais. Então temos que $\sigma = [v_1, v_2, ..., v_n]$ constitui um esquema de eliminação perfeito. Sabemos pelo Lema 3.4, que um grafo é cordal se, e somente se, possui tal esquema. Portanto a k-árvore T_n é cordal. ■

Considere como exemplo para o Lema 3.5 a Figura 3.6(b) e o esquema de eliminação $\sigma = [v_8, v_7, v_6, v_5, v_4, v_3, v_2, v_1]$.

O Lema 3.6 a seguir prova que uma k-árvore possui largura limitada em k. Isso se deve ao fato dessa classe ser subclasse dos grafos cordais, e esta classe, por sua vez, admite DEA cujas bolsas possuem somente cliques.

Lema 3.6 *[30] Uma k-árvore tem largura em árvore k.*

Prova. Pelo Lema 3.5 sabemos que uma k-árvore é um grafo cordal, e, pelo Teorema 3.1, admite uma decomposição em árvore mínima onde toda parte é uma clique maximal. Todas as cliques da k-árvore tem tamanho $k+1$. Portanto, uma k-árvore tem largura k. ■

Lema 3.7 *[18] Se G é um grafo cordal com pelo menos $k+1$ vértices e tem $\omega(G) = k+1$, então G pode ser imerso em uma k-árvore.*

Prova. Seja $\sigma = [v_1, v_2, ..., v_n]$ um esquema de eliminação perfeito de G. Sabemos que σ existe devido ao Lema 3.4. Fazemos uma *imersão* de G em uma k-árvore como a seguir:

- Primeiro adicionaremos arestas tal que o subgrafo induzido por $\{v_{n-k}, ..., v_n\}$ torne-se uma $(k+1)$-clique;
- Para o passo da indução, assumimos que o subgrafo com os vértices $v_{i+1}, ..., v_n$ foi triangularizado em uma k-árvore T_{i+1}. Em G o vértice v_i é adjacente à clique C com no máximo k vértices em $\{v_{i+1}, ..., v_n\}$. Na k-árvore T_{i+1}, C está contida em uma k-clique C'. Faça v_i não adjacente a k-vértices desta clique. Obtemos, assim, uma k-árvore T_i que é uma imersão de G em uma k-árvore.

■

Veja um exemplo de uma imersão de um grafo $G = (V, E)$ na Figura 3.9. O passo a passo dessa imersão é dado a seguir.

- **1º Passo**: $\{v_{n-k}, ..., v_n\} = \{v_6, v_7, v_8, v_9\}$. Adicionamos arestas para obtermos uma $(k+1)$-clique=4-clique.

- **Passo de indução**: Consideramos que $\{v_{i+1}, ..., v_n\}$ já foi triangularizado. Observe que o vértice v_i representa o novo vértice que é inserido na árvore, pela definição de k-árvore. Vá selecionando os vértices v_i (i em ordem decrescente a partir do vértice v_{n-k-1}) e faça-o adjacente a qualquer k-clique do grafo atual.
  - Selecione o vértice v_5 e faça-o adjacente à 3-clique $C' = \{v_6, v_7, v_8\}$.
  - Selecione o vértice v_4 e faça-o adjacente à 3-clique $C' = \{v_5, v_6, v_7\}$.
  - Selecione o vértice v_3 e faça-o adjacente à 3-clique $C' = \{v_4, v_5, v_6\}$.
  - Selecione o vértice v_2 e faça-o adjacente à 3-clique $C' = \{v_3, v_4, v_5\}$.
  - Selecione o vértice v_1 e faça-o adjacente à 3-clique $C' = \{v_2, v_3, v_4\}$.

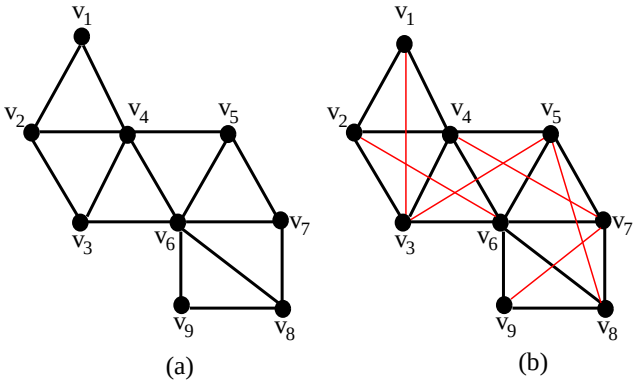

Figura 3.9: (a) Grafo a ser imerso em uma 3-árvore; (b) Grafo já imerso em uma 3-árvore

Teorema 3.2 *[4] As três declarações seguintes são equivalentes:*

(i) G é uma k-árvore parcial.

(ii) G é subgrafo de um grafo cordal G' com $\omega(G') \leq k+1$.

(ii) G é um subgrafo de um grafo cordal G' com $\omega(G') = k+1$.

Prova. $(i) \rightarrow (ii)$: Segue diretamente da definição de k-árvore parcial, sabendo que uma k-árvore é um grafo cordal e que a clique máxima tem tamanho no máximo $k+1$ pela definição de k-árvore.

$(ii) \rightarrow (iii)$: É trivial para se G' tem $\omega(G') < k+1$. Basta adicionar arestas a G' até a clique máxima ter tamanho $k+1$. Observe que G continuou a ser subgrafo de G'.

$(iii) \rightarrow (i)$: Ocorre devido ao Lema 3.7. Podemos adicionar arestas em G' para obtermos uma k-árvore. Assim, qualquer subgrafo de G' é uma k-árvore parcial. ■

Teorema 3.3 *[30] G é uma k-árvore parcial se, e somente se, G tem largura em árvore no máximo k.*

Prova. ($\Rightarrow$) Vamos provar que toda k-árvore parcial tem largura no máximo k. Iremos mostrar um resultado mais forte, onde k-árvore tem largura no máximo k. Como toda k-árvore parcial é um subgrafo de uma k-árvore, e a largura de um grafo é pelo menos igual a largura de seus subgrafos (como já provamos na Propriedade 2.1), o resultado segue nessa direção.

Seja $G = \big(V, E\big)$ uma k-árvore com $\big|V\big| > k+1$. Existe um vértice $v \in V$ tal que $G\big[V - \{v\}\big]$ é uma k-árvore, e os vizinhos de v induzem uma clique K de tamanho k em G. Usando indução, nós podemos assumir que $G\big[V - \{v\}\big]$ tem largura no máximo k com uma decomposição em árvore correspondente $\Big(\{X_i | i \in I\}, T = \big(I, M\big)\Big)$. (O caso base da indução é quando $G\big[V - \{v\}\big]$ é um grafo completo com k vértices; tal grafo tem claramente largura $k-1$). Pelo Lema 2.3, existe um $i' \in I$ com $X_{i'}$ contendo todos os vizinhos de v em G, com $K \subseteq X_{i'}$. Seja $J = I \cup \{j\}$ onde $j \notin I$ e seja $X_j = K \cup \{v\}$. Agora $\Big(\{X_i | i \in J\}, T = \big(J, M \cup \{i, j\}\big)\Big)$ é uma decomposição em árvore de G com largura k.

($\Leftarrow$) Nessa direção, queremos mostrar que todo grafo com largura no máximo k é uma k-árvore parcial. Sejam $G = \big(V, E\big)$ um grafo com $|V| \geq k+1$ e $D = \Big(\{X_i | i \in I\}, T = \big(I, M\big)\Big)$ a decomposição em árvore de G com largura no máximo k. Provaremos, por indução em $|V|$, que existe uma k-árvore $H = \big(V, E'\big)$ tal que, para todo $i \in I$, $G[X_i]$ é subgrafo de uma clique de H com $k+1$ vértices. Se $|V| = k+1$ então a prova está concluída. Caso contrário, tome um nó folha $l \in I$ e seja $j \in I$ o único vizinho de l. Se $X_l \subseteq X_j$, então podemos remover l de T e continuarmos com a decomposição restante. Seja v o único vértice em $X_l - X_j$ (caso contrário X_l pode ser dividido em alguns nós tal que o novo nó folha resultante satisfaça essa exigência). Agora é importante notar as seguintes observações: considerando que v não aparece em qualquer X_i para $i \neq l$, todos os vizinhos de v devem aparecer em X_l. Considerando que $|X_l| \leq k+1$, v tem no máximo k vizinhos. Também, sendo v o único vértice em $X_l - X_j$ todos os vizinhos de v devem também aparecer em X_j. Suponha que a hipótese de indução em $G[V - \{v\}]$ com decomposição em árvore $D' = \big(\chi - X_l, T = (I - \{l\}, F'\big)$ resulta em uma k-ávore H. Lembre-se que v tem no máximo k vizinhos em G e eles são todos pertencentes a X_j. Portanto, os vizinhos de v induzem um subgrafo de uma k-clique C de H em G. Agora, podemos adicionar a H, v e suas arestas adjacentes a todos os vértices de C (e faremos uma $(k+1)-$clique) e teremos a k-árvore desejada. ■

O próximo Lema estima o número máximo de arestas de um grafo G, se este possui largura em árvore limitada.

Lema 3.8 *[6] Se a largura em árvore de $G = \big(V, E\big)$ é no máximo k, então $|E| \leq k|V| - \frac{1}{2}k(k+1)$.*

Prova. Seja $G' = \big(V, E'\big)$ uma k-árvore. Sabemos pelo Lema 3.5 que uma k-árvore é um grafo cordal, então G' admite uma DEA onde toda bolsa é uma clique (Teorema 3.1). Por definição, todas as cliques de uma k-árvore tem tamanho $k+1$. É fácil ver usando indução que $|I| = |V| - k$, já que para quaisquer X_i e X_j tais que $|X_i \cap X_j| \neq \emptyset$, tem-se $|X_i \cap X_j| = k$. Se $|V| = k+1$, então T tem somente um nó e $|E'| = \frac{1}{2}k(k+1)$. Se T tem I nós, então foram acrescentados $I-1$ vértices a K_{k+1}, feitos adjacentes a k vértices, de modo a obter G' pela definição de k-árvore. Vale notar que, para cada vértice acrescentado, foram adicionadas k arestas. Desta forma, todo vértice de G' tem no mínimo k vizinhos - contribuindo com $k|V|/2$ arestas - e, considerando-se as novas arestas adicionadas ao longo da construção de G', tem-se uma contribuição de $k(|V| - k - 1)/2$ arestas. Logo,

$$|E'| = \tfrac{k.|V|}{2} + \tfrac{k.(|V|-k-1)}{2} = k.|V| - \tfrac{1}{2}k(k+1).$$

Se G tem largura em árvore no máximo k, então, pelo Teorema 3.3, G é uma k-árvore parcial. Assim, $|E| \leq |E'| = k.|V| - \frac{1}{2}k(k+1)$. ■

Proposição 3.1 *Seja $G = \big(V, E\big)$ uma k-árvore parcial com k minimal. Então $min_H\big(\omega(H) - 1\big) = k$ onde H é uma imersão de G.*

Prova. Suponha que G é uma k-árvore parcial com k minimal. Pelo Teorema 3.2, sabemos que G é um subgrafo de uma grafo cordal G' com $\omega(G') = k+1$. Assim, existe uma imersão G' de G com $\omega(G') = k+1$. Suponha que a menor imersão possível não seja G'. Então existe uma imersão H com $\omega(H) = k'' < k+1$. Como H é cordal e $\omega(H) = k''$, pelo Lema 3.7 podemos adicionar arestas a H tal que o grafo resultante seja uma $(k''-1)$-árvore. Então G deveria ser uma $(k''-1)-$árvore com $k''-1 < k$. Isso é uma contradição com nossa hipótese de que k é minimal. Assim, não existe uma imersão H de G com $\omega(H) < k+1$ e então $min_H\big(\omega(H) - 1\big)$ sobre todas as triangularizações H de G. ■

Proposição 3.2 *Seja* $G = \big(V, E\big)$ *um grafo e* $min_H\big(\omega(H) - 1\big) = k$, *onde* H *é uma triangularização de* G. *Então* G *tem largura* k.

Prova. Seja H a menor triangularização de G com $\omega(H) = k+1$. Como H é cordal, pelo Corolário 3.1, $LA(H) = \omega(H)-1 \Rightarrow LA(H) = k+1-1 \Rightarrow LA(H) = k$. Além disso, como G é um grafo parcial de H, pela Propriedade 2.1. $LA(G) \leq LA(H) = k$. Agora suponha que $LA(G) < k$, isto é, $LA(G) = k' < k$. Então G é uma k'-árvore parcial (Teorema 3.3), k' minimal, e assim $min_H\big(\omega(H) - 1\big) = k' < k$. onde H é uma triangularização de G. Isso é uma contradição. E assim concluímos que G tem largura k. ■

3.2.3 Algoritmo de decomposição em árvore para *k*-árvores

Nesta seção, apresentamos um algoritmo de DEA para k-árvores, baseado em seu esquema de eliminação perfeito.

Uma k-árvore H tem um esquema de eliminação perfeito $\sigma = \big[v_1, ..., v_n\big]$ tal que para cada i, $1 \leq i \leq n-k$, o conjunto de vértices $B_i = \{v_i\} \cup \big(N(v_i) \cap \{v_{i+1}, ..., v_n\}\big)$ induz uma $(k+1)$-clique em H.

Chamaremos a DEA de uma k-árvore de *árvore-EEP baseada em* σ.

A construção da árvore-EEP P de G baseada em σ é uma árvore enraizada com nós $V\big(P\big) = \{B_1, ..., B_{n-k}\}$ e definida como a seguir:

→ O nó B_{n-k} é a raiz de P;

→ Um nó B_i, $1 \leq i < n-k$, tem como seu pai o nó B_j, $1 \leq j \leq n-k$, tal que j é a bolsa de menor índice tal que $\big|B_i \cap B_j\big| = k$ (note que essa interseção não contém v_i).

A árvore-EEP P é uma DEA de largura k de H.

Na Figura 3.10 temos um exemplo de uma imersão em uma k-árvore e sua DEA construída como explicado anteriormente.

Para sabermos quais são os pais e os filhos da DEA temos que testar as interseções como a seguir:

→ $B_1 \cap B_2 = \{6\}$

→ $B_1 \cap B_3 = \{6\}$

→ $B_1 \cap B_4 = \{9\}$

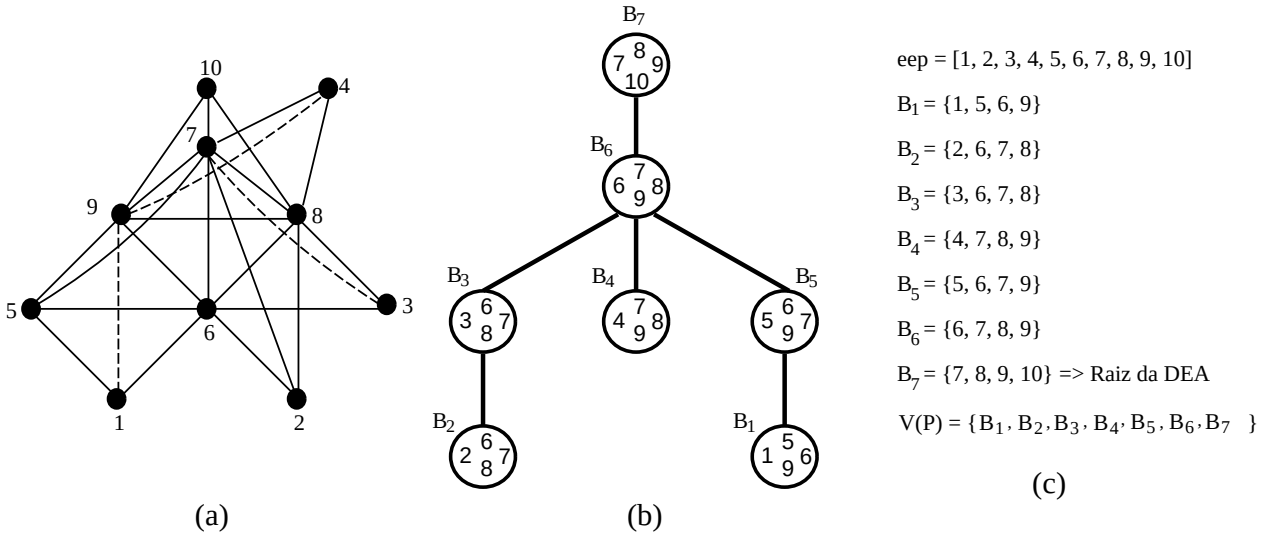

Figura 3.10: (a) Uma 3-árvore parcial G imersa em uma 3-tree H; (b) Sua árvore-EEP P; (c) Vértices de P

$\rightarrow B_1 \cap B_5 = \{5, 6, 9\}$: B_5 é o pai de B_1

$\rightarrow B_2 \cap B_3 = \{6, 7, 8\}$: B_3 é o pai de B_2

$\rightarrow B_3 \cap B_4 = \{7, 8\}$

$\rightarrow B_3 \cap B_5 = \{6, 7\}$

$\rightarrow B_3 \cap B_6 = \{6, 7, 8\}$: B_6 é o pai de B_3

$\rightarrow B_4 \cap B_5 = \{7, 9\}$

$\rightarrow B_4 \cap B_6 = \{7, 8, 9\}$: B_6 é o pai de B_4

$\rightarrow B_5 \cap B_6 = \{6, 7, 9\}$: B_6 é o pai de B_5

$\rightarrow B_6 \cap B_7 = \{7, 8, 9\}$: B_7 é o pai de B_6

Uma observação importante a se fazer é que qualquer grafo G é uma k-árvore parcial para algum inteiro k. E, como foi mencionado anteriormente, qualquer k-árvore parcial está imersa numa k-árvore, cuja largura é k. Portanto, uma questão interessante é dado um grafo G qualquer, encontrar uma k-árvore H, $G \subseteq H$ com k mínimo, uma vez que k será um limite superior para a largura de G.

3.3 Heurísticas de triangularização

Como já vimos anteriormente, a largura em árvore de um grafo G é igual a largura mínima sobre todas as decomposições em árvore de G. Equivalentemente, a largura em árvore de G é o mínimo $k \geq 0$ tal que G é um subgrafo de um grafo cordal com todas as cliques de tamanho no máximo $k+1$. Assim, qualquer triangularização do grafo define uma decomposição em árvore de um grafo com a mesma largura em árvore. Similarmente, toda decomposição em árvore de um grafo define uma triangularização de G com a mesma largura em árvore. Como encontrar uma DEA para um grafo cordal G depende da identificação de todas as cliques maximais de G, o que pode ser feito em tempo linear usando um esquema de eliminação perfeito, é interessante conhecermos formas de triangularizar grafos.

Nesta seção veremos um algoritmo de triangularização minimal com tempo de execução $O(nm)$.

3.3.1 Lemas, Teoremas e Corolários

Embora o conceito de DEA ainda não tivesse sido introduzido por Gavril no Teorema 3.4 a seguir, este conceito foi usado para caracterizar grafos cordais.

Teorema 3.4 *[14] Um grafo $G = \big(V, E\big)$ é cordal se, e somente se, G pode ser construído como o grafo interseção de subárvores de árvores, isto é, existe uma árvore $T = \big(I, F\big)$ tal que podemos associar uma subárvore $T_v = \big(I_v, F_v\big)$ de T com cada vértice $v \in V$, tal que $(v, w) \in E$ se, e somente se, $I_v \cap I_w \neq \emptyset$.*

Prova. Seja $\{T_v\}$, $v \in V$, uma família de subárvores de uma árvore T tal que $(v, w) \in E$ se, e somente se, $T_v \cap T_w \neq \emptyset$. Suponha que G contém um ciclo sem corda $\big[v_0, v_1, ..., v_{k-1}, v_0\big]$ com $k > 3$ correspodendo a uma seqüência de subárvores $T_0, T_1, ..., T_{k-1}$ da árvore T, isto é, $T_i \cap T_j \neq \emptyset$ se, e somente se, i e j diferem de no máximo 1 mod k. Toda aritmética será feita no módulo k. Escolha um ponto a_i de $T_i \cap T_{i+1}$, $\big(i = 0, ..., k-1\big)$. Seja b_i um ponto em comum no (único) caminho simples de a_i a a_{i-1} e a_i a a_{i+1}. Esses caminhos se encontram em T_i e T_{i+1} respectivamente, então b_i também se encontra em $T_i \cap T_{i+1}$. Seja P_{i+1} o único caminho simples conectando b_i e b_{i+1}. Claramente, $P_i \subseteq T_i$, então $P_i \cap P_j = \emptyset$ para i e j diferindo de mais de 1 mod k. Além disso, $P_i \cap P_{i+1} = b_i$ para $i = 0, ..., k-1$. Assim, $\bigcup_i P_i$ é um ciclo simples em T, contradizendo a definição de uma árvore. ■

O Lema a seguir mostra que é possível triangularizar qualquer grafo de forma que a largura em árvore não aumente. Assim, obtendo uma boa triangularização, poderemos facilmente encontrar a largura de grafos em geral.

Lema 3.9 *[19] Para todo grafo* $G = \big(V, E\big)$ *existe uma triangularização* G' *de* G, $G' = \big(V, E' = E \cup E_t\big)$, *com* $LA\big(G'\big) = LA\big(G\big)$.

Prova. Seja G um grafo geral e $\big(\chi, T = (I, F)\big)$ uma decomposição em árvore de largura mínima. Construíremos um grafo $G' = \big(V, E'\big)$ pela seguinte regra: $\big(v, w \in E'\big)$ se, e somente se, $\big(v, w \in X_i\big)$ para algum $i \in I$. Pelo Teorema 3.4 é claro que G' é cordal. Pela segunda propriedade de decomposição em árvore, o conjunto de arestas E' pode ser dividido em duas partes E e E_t. Então, G' é uma triangularização de G pela adição das arestas de triangularização E_t. Além disso, a largura de G e G' é igual, $LA\big(G'\big) = LA\big(G\big)$. ■

Como conseqüência direta do Lema 3.9, temos o seguinte Corolário.

Corolário 3.2 *Encontrar a largura em árvore de um grafo* G *é equivalente a encontrar uma triangularização* G' *de* G *com clique de tamanho mínimo.*

O algoritmo Lex_M mostrado a seguir foi introduzido por Rose et al [27]. Este foi um dos primeiros algoritmos conhecidos que produz uma triangularização minimal G' de um grafo $G = \big(V, E\big)$.

Algoritmo 3.2 *DEA_Cordal - Algoritmo de decomposição em árvore mínima para grafos cordais*

- *Algoritmo: Lex_M*
- *Entrada: Um grafo* $G = \big(V, E\big)$
- *Saída: Um esquema perfeito de eliminação* σ *e uma triangularização minimal* G' *de* G

1. **Inicio**
2. $G' = G$

3. $\forall u \in V(G)$ **faça** $L(u) = \emptyset$

4. **para** $i = n$ **até** *1* **faça**

5. *Escolha o vértice não numerado v de maior rótulo $L(v)$*

6. $\sigma = \sigma + \{v\}$

7. **para** *cada vértice não numerado w: $\exists$ caminho $v = v_1, v_2, ..., v_{k+1} = w$ com v_j não numerado e $L(v_j) <_{lex} L(w)$, $j = 2, 3, ..., k$* **faça**

8. *Adicione i a $L(w)$*

9. *Adicione a aresta de preenchimento (v, w) a G'*

10. **fim para**

11. **fim para**

12. **Fim**

Um algoritmo de triangularização é interessante se este produzir uma triangularização minimal para não aumentar a complexidade da execução na hora da resolução dos problemas. O Lema a seguir prova exatamente isso, ou seja, que o algoritmo Lex_M que estamos estudando produz uma triangularização desse tipo.

Teorema 3.5 *[27] O algoritmo Lex_M produz uma triangularização minimal G' de um grafo G.*

Prova.

Iremos discutir algumas das propriedades do algoritmo Lex_M. A primeira observação é que os vértices numerados sempre formam um subgrafo conexo de G. Para explicar isso observe quando o primeiro número é adicionado ao rótulo L_u de um vértice u. Assuma que o número do vértice v é o primeiro número adicionado a L_u, quando existe um caminho de menores rótulos de v a u no grafo. Como $L_u = \emptyset$ antes do número de v ser adicionado, então segue que uma aresta (v, u) é o único caminho possível de v a u. Como conseqüência disso, G' é um supergrafo de G considerando que toda aresta em $E(G)$ é considerada como o menor caminho numerado entre todos os vértices de G (isto é, o número de arestas nunca diminuem). Segue disso que o conjunto de vértices não numerados com um rótulo não vazio é exatamente a vizinhança dos vértices numerados.

Podemos concluir que existe um caminho entre os vértices numerados e todo par de vértices que tem um rótulo não vazio, e podemos também concluir que o algoritmo Lex_M somente introduz uma aresta (u, v) se u é o vértice com maior rótulo e $L_v \neq \emptyset$. Uma aresta (u, v) é representada em G' como o número de u no rótulo de v se $L_u \geqslant L_v$, e sabemos que a aresta (u, v) é representada no grafo de entrada se $L_v = \emptyset$.

Como consequência das observações acima, o algoritmo ao introduzir uma aresta (u, v) faz existir dois caminhos disjuntos de u a v. Então toda aresta (u, v) introduzida pelo Lex_M é uma corda de algum ciclo de G. ■

O algoritmo construído dessa forma executa em tempo $O(nm)$, pois para cada vértice $v \in V(G)$ são analisados seus vizinhos, uma vez para cada vizinho.

3.3.2 Exemplo de triangularização

Aqui será apresentada passo a passo a execução do algoritmo no grafo G da Figura 3.11.

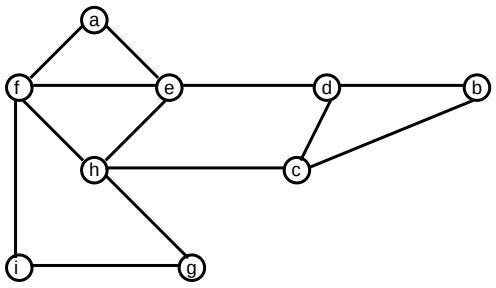

Figura 3.11: Grafo G a ser triangularizado

No passo 3 do algoritmo, todos os vértices são rotulados com $\emptyset$ (Figura 3.12(a)). Neste momento, o esquema de eliminação perfeito ainda se encontra vazio.

No passo 5 (Figura 3.12(b)), escolhemos o vértice não numerado de maior rótulo $L(v)$. Neste momento, todos os vértices encontram-se não-numerados e com o mesmo rótulo $\emptyset$. Podemos, portanto, escolher qualquer um dos vértices dentre $a, b, ..., h, i$. Escolhemos então, o vértice i. O vértice i é inserido no esquema de eliminação perfeito *eep*.

Nos passos 7 a 10, i é adicionado ao rótulo de todo vértice w para o qual existe caminho entre i e w, onde todos os vértices no interior do caminho são não-numerados e tal que seus rótulos são menores do que o rótulo de w. Para melhor visualização da execução do algoritmo, nós adicionamos um valor numérico aos rótulos, cada valor referente a um

vértice. Neste caso, adicionamos 8 ao rótulo de todo vértice w referido anteriormente. Na Figura 3.12(b) a única aresta a ser adicionada é a aresta (h, i).

A Figura 3.12(c) representa a execução do algoritmo quando o vértice h é escolhido no passo 5. Neste caso, os vértices c, d, e, f, e g são rotulados 7 e a aresta (d, h) é adicionada a G'.

Neste momento, voltamos ao passo 5 e dois vértices possuem maior rótulo igual a 87. Escolhemos o vértice g e o inserimos no esquema de eliminação perfeito. Como podemos facilmente ver na Figura 3.12(d), não existe nenhum caminho como descrito nos passos 7 a 9, então nenhum vértice tem seu rótulo modificado e nenhuma aresta é adicionada.

Novamente no passo 5, escolhemos então o vértice f de valor numérico 5. Nos passos 7 a 10 rotulamos o vértice a e o vértice e, que são vértices adjacentes a f. Observe que ao rotularmos o vértice e não existe mais caminho entre f e os outros vértices. Assim, nenhuma outra aresta é adicionada. Veja Figura 3.12(e).

Outra vez no passo 5, o vértice de maior rótulo escolhido é o vértice e de rótulo 75. Nos passos 7 a 10, rótulamos os vértices a e d (vértices adjacentes a e) com valor 4. Observe que ao escolhermos rotular primeiro o vértice d, não é possível chegar aos vértices c e b por nenhum caminho. Então o passo 11 não é executado. (Figura 3.12(f)).

A Figura 3.12(g) refere-se aos passos 5 a 10 em que o vértice d é numerado e os vértices c e b são rotulados com 3. Nenhuma aresta é adicionada.

Neste momento (3.12(h)) restam três vértices ainda não numerados: os vértices a, b e c, com os respectivos rótulos, 54, 32 e 73. Assim, no passo 5 do algoritmo, escolhemos o vértice c. E o único caminho possível verificados nos passos 7 a 9 é o caminho c, b. Como esse caminho é uma aresta nenhuma aresta é adicionada no passo 11.

Na Figura 3.12(i), o passo 5 é executado escolhendo o vértice a. Nos passos 7 a 9 nenhum caminho é encontrado, e portanto nenhuma aresta é adicionada.

E finalmente, o último vértice é escolhido na Figura 3.12(j). Então b é numerado, e como não existe nenhum vértice não-numerado, nenhuma aresta é adicionada.

Como já foi mencionado, decomposições em árvores facilitam a resolução de muitos problemas combinatórios. No entanto, torna-se mais fácil resolver tais problemas se a decomposição em árvore for, o que chamamos de, uma boa decomposição em árvore. A seção a seguir define uma boa decomposição em árvore e mostra como transformar uma decomposição em árvore qualquer numa boa em tempo linear.

3.4 Boa decomposição em árvore

Uma DEA pode ser facilmente convertida em tempo linear numa DEA especial com a mesma largura em árvore, e com número de nós no máximo quatro vezes o número de nós da DEA [18]. Tal decomposição será chamada de *boa decomposição em árvore* (boa-DEA).

Uma boa-DEA $\big(\chi = \{X_i | i \in T\}, T = (I, F)\big)$ é uma árvore binária e enraizada e seus nós são de quatro tipos:

- *Nó Folha* i : são folhas de T e tem $|X_i| = 1$;
- *Nó Introduz* i : tem um filho j com $X_i = X_j \cup \{v\}$ para algum vértice $v \in V$;
- *Nó Esquece* i : tem um filho j com $X_i = X_j - \{v\}$ para algum vértice $v \in V$;
- *Nó Junção* i : tem dois filhos j_1 e j_2, com $X_i = X_{j_1} = X_{j_2}$.

Fazendo uma comparação entre boas-DEAs e DEAs, em geral, a boa-DEA não aumenta as possibilidades algorítmicas, mas o projeto de algoritmos é considerado mais fácil e podemos em muitos casos obter melhores fatores constantes no tempo de execução dos algoritmos.

O Lema 3.10 mostra que toda k-árvore tem um boa-DEA de largura k, e de que forma podemos construí-la.

Lema 3.10 *[18] Todo grafo com largura em árvore k tem uma boa-DEA de largura k. Além disso, se n é o número de vértices de G então existe uma boa-DEA com no máximo $4n$ nós.*

Prova. Considerando que G tem largura em árvore k, G tem pelo menos $k+1$ vértices e existe uma triangularização de G em uma k-árvore H 3.7. Mostraremos que existe uma boa-DEA para H de largura k. Seja n o número de vértices de H. Se $n = k+1$, podemos tomar a DEA trivial que tem um único nó e o subconjunto correspondente conterá todos os vértices. Assuma que $n > k+1$. Então H tem um vértice simplicial x. Seja H' a k-árvore obtida de H pela remoção de x. Por indução, existe uma boa-DEA $(\{X_i | i \in I\}, T = (I, F))$ para H' com no máximo $4(n-1)$ nós. Seja C o conjunto de vizinhos de x. Considerando que C é uma clique (com k vértices) existe um subconjunto X_i na boa-DEA contendo C. Chame-o de nó i. Considere três casos:

Caso 1: Primeiro assuma que i tem dois filhos j e k. Então $X_i = X_j = X_k$. Então desça na árvore até um dos filhos. E continue até encontrar um nó p com no máximo um filho. Faça $i = p$.

Caso 2: Se i é um nó folha, adicione um ou dois nós posteriores a i da seguinte maneira. Se $X_i = C$ então tome um novo nó $a \notin I$ e um subconjunto $X_a = C \cup \{x\}$ e faça-o um filho de i. Caso contrário, se $X_i \neq C$, seja $z \in X_i - \{C\}$. Então dê a i um novo filho a e crie um subconjunto $X_a = X_i - \{z\}$. Note que considerando $C \subset X_i$ e $k = |C| \leq |X_i| \leq k + 1$, $X_a = C$. Tome um outro nó b e um subconjunto $X_b = C \cup \{x\}$ e faça b um filho de a. Então o número de novos vértices introduzidos quando i é uma folha é no máximo dois.

Caso 3: Finalmente assuma que i tem um único filho q. Remova a aresta (i, q) da árvore. Tome um nó p e seu subconjunto correspondente $X_p = X_i$. Faça p um filho de i e q um filho de p. Tome outro novo nó b, faça um novo subconjunto $X_b = X_i$ e faça b um filho de i na árvore (assim, i tem agora dois filhos p e b). Agora podemos descer na árvore para b que é uma folha, e continuamos com descrito no *Caso 2*. Note que nesse caso nós introduzimos no máximo 4 novos nós.

Considerando que a DEA para H' tem no máximo $4(n-1)$ nós, a DEA de G tem no máximo $4n$ nós. ■

Em nosso exemplo da Figura 3.14, utilizamos o grafo da Figura 3.13(a) que foi triangularizado com o algoritmo de triangularização de [27] apresentado na seção 3.3 mas para efeito didático, não foi utilizado o esquema perfeito de eliminação gerado pelo algoritmo, e sim $\sigma = \{a, g, i, b, c, d, e, f, h\}$, de forma que todos os casos apresentados no Lema 3.10 fossem utilizados.

Para construir uma boa-DEA, primeiro precisamos decompor o grafo em subgrafos até que restem $n = k + 1$ vértices. Isso é feito eliminando os vértices do esquema perfeito de eliminação, como foi feito na Figura 3.13(a) a Figura 3.13(g) Podemos então construir a boa-DEA, e segue a seguir uma descrição dessa construção.

Começamos a construir a boa-DEA a partir do subgrafo da Figura 3.13(g). Temos o primeiro nó da DEA mostrado na Figura 3.14(a).

O vértice que temos que inserir na DEA é o vértice d, e o subgrafo montado com a inserção de d pode ser visto na Figura 3.13(f). O conjunto de vizinhos de d é $C = \{e, h\}$. Como o único vértice da boa-DEA é o vértice raiz, podemos aplicar o Caso 2 do Lema

3.10. Como $X_i \neq C$, dois nós são inseridos na Boa-DEA, tendo como resultado a árvore da Figura 3.14(b).

Ao inserirmos o vértice c formamos o subgrafo da Figura 3.13(f). Temos agora a vizinhança $C = \{d, h\}$. Novamente caímos no Caso 2. A DEA para esse subgrafo pode ser vista na Figura 3.14(c). Novamente $X_i \neq C$ então inserimos dois novos nós.

Agora com a inserção do vértice b formamos o subgrafo da Figura 3.13(d) e temos $C = \{d, c\}$. A nossa opção é o Caso 2 e a DEA montada pode ser vista na Figura 3.14(d).

Com a inserção do vértice i construímos o subgrafo da Figura 3.13(c) e podemos visualizar na Figura 3.14(e) o funcionamento do Caso 3. A vizinhança de i é $C = \{f, h\}$. Os vértices da vizinhança encontram-se na bolsa do nó raiz, que até esse momento possuía um único filho. Assim, inserimos dois nós no Caso 3, e mais dois nós devido ao motivo do Caso 3 chamar o Caso 2. Sendo assim, 4 novos nós foram inseridos.

Vamos agora inserir o vértice g que tem como vizinhos os vértices h e i. Voltamos ao Caso 2. Construímos a DEA da Figura 3.14(f).

E, finalmente, inserimos o vértice a, montando então o grafo G original, e podendo assim montar a boa-DEA do grafo. Os vizinhos de a são os vértices e e f, que estão alocados na bolsa do nó raiz i, e que neste momento possui dois filhos. Podemos, portanto, visualizar o funcionamento do Caso 1. Neste momento devemos descer na árvore até encontrarmos um nó p com um único filho. Podemos escolher qualquer um dos lados direito ou esquerdo para descer. Neste exemplo o lado esquerdo foi o escolhido e p é exatamente o filho direito de i. Encontrado o nó p devemos aplicar o Caso 3 que também utiliza o Caso 2, e quatro novos nós foram inseridos, como mostra a Figura 3.14g.

Assim, o Lema a seguir prova que possível construir uma boa-DEA a partir de uma DEA qualquer de largura k sem aumentar sua largura.

Lema 3.11 *[18] Para k constante, dada uma DEA de um grafo G de largura k e $O(n)$ nós, onde n é o número de vértices de G, nós podemos encontrar uma boa-DEA de G de largura k e com no máximo $4n$ nós em tempo $O(n)$.*

Prova. A construção do grafo cordal implicado pela DEA claramente toma tempo linear (veja Teorema 3.1, página 23). Dado um grafo cordal, podemos encontrar um esquema perfeito de eliminação (Veja Lema 3.4). O Lema segue agora facilmente da prova construtiva do Lema 3.10. ■

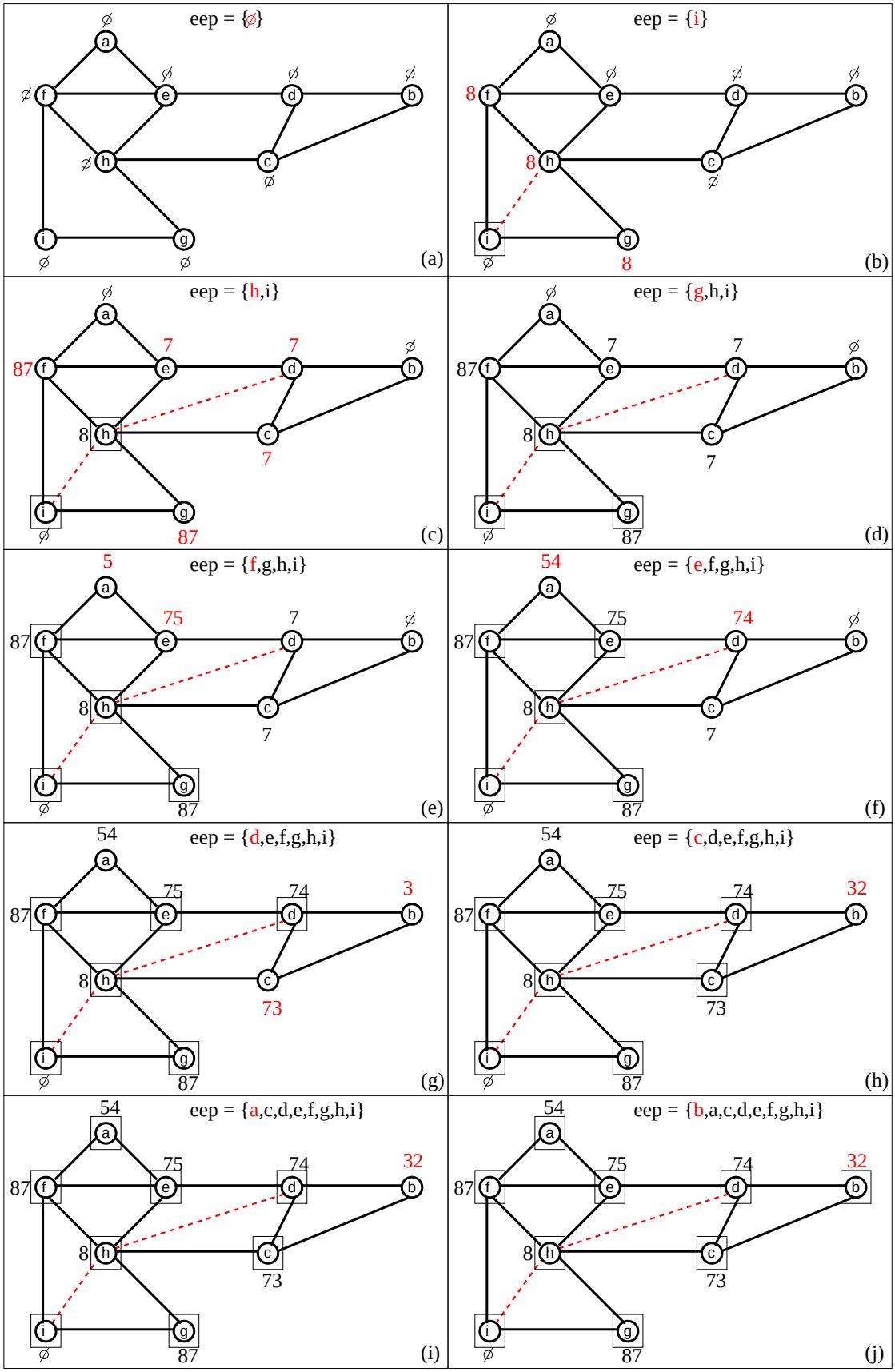

Figura 3.12: Uma execução do algoritmo Lex_M

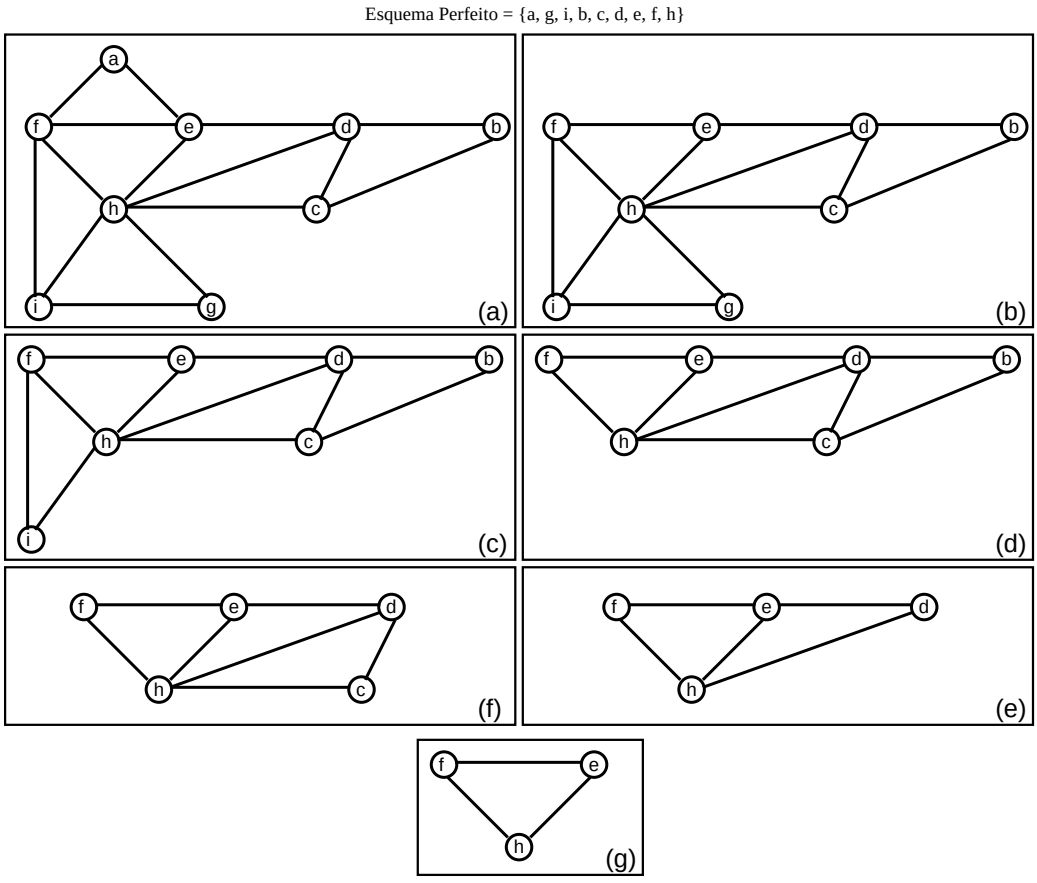

Figura 3.13: Decomposição do grafo para a construção da boa-DEA

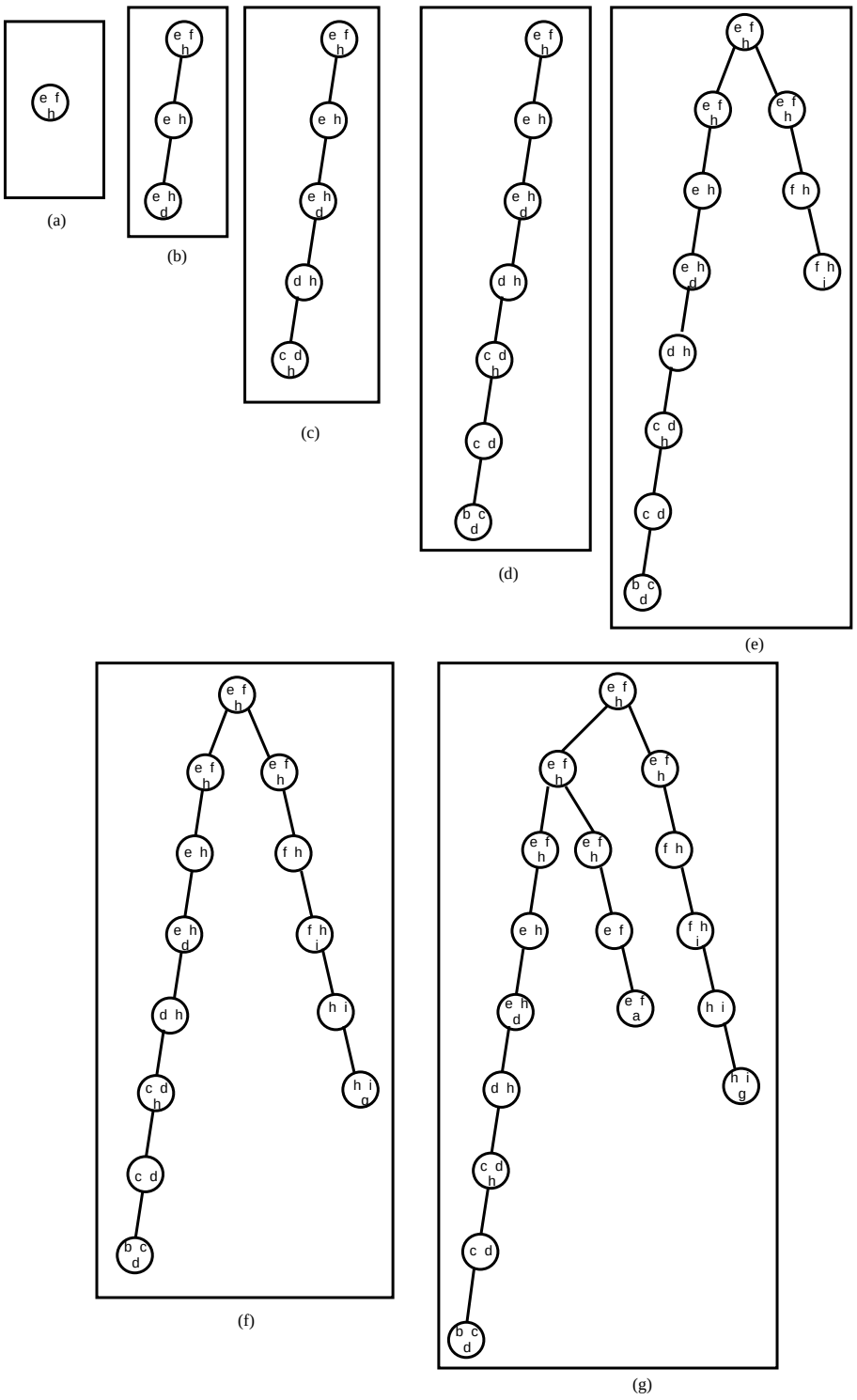

Figura 3.14: Construindo uma boa-DEA

Capítulo 4

Como resolver problemas NP-Difíceis com a técnica de DEA

Neste capítulo serão ilustrados algoritmos para resolver problemas NP-Difíceis em grafos restritos a k-árvores parciais. Abordaremos dois problemas: problema de otimização envolvendo conjuntos independentes e problema de coloração com listas.

O *problema de conjunto independente (PCI)* é um dos problemas fundamentais em otimização combinatória. Um conjunto independente em um grafo é um conjunto de vértices que são mutuamente não adjacentes. O problema de conjunto independente trata em encontrar o conjunto independente de tamanho máximo em um dado grafo. O *problema do conjunto independente ponderado máximo(PCIP)* consiste em encontrar o conjunto independente de peso máximo em um grafo cujos vértices possuem peso.

O *problema de coloração de grafos* se resume em colorir os vértices de um grafo de forma que a coloração seja própria, isto é, vértices adjacentes não possuem a mesma cor. A *coloração por lista* é um tipo de coloração de grafo especial onde cada vértice pode estar restrito a uma lista de cores permitidas.

Esses problemas são conhecidamente NP-Difíceis para grafos em geral, mas podem ser resolvidos em tempo linear para árvores. Então, a utilização de DEA se torna muito importante.

4.1 Problema do conjunto independente ponderado

Dado um grafo $G = \big(V, E\big)$ com pesos $c(v) \in \mathbb{Z}^+$ para cada vértice $v \in V$, o objetivo é encontrar um subconjunto S de vértices mutuamente não adjacentes e cuja soma dos

pesos $c(s) = \sum_{v \in S} c(v)$ é maximizado.

Como exemplo de conjunto independente e conjunto independente ponderado, considere o grafo da Figura 4.1. Nesse caso, estamos considerando que todos os vértices possuem o mesmo peso $c(v) = 1$.

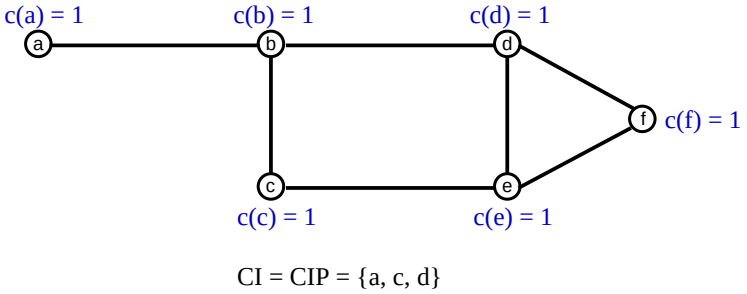

Figura 4.1: Exemplo de conjunto independente e conjunto independente ponderado

4.2 Problema do conjunto independente ponderado em árvores

Para árvores, os problemas PCI e PCIP podem ser resolvidos em tempo linear. Veja como podemos resolver o PCIP: escolha um vértice como raiz v e denote $T(v)$ como a subárvore com v como raiz. Denote $A(v)$ o conjunto independente de peso máximo em $T(v)$, e $B(v)$ o conjunto independente de peso máximo em $T(v)$, não contendo v. Com essa construção, teremos em $A(r)$ ou $B(r)$ o CIP para o grafo G.

$A(v)$ e $B(v)$ são calculados de maneira bottom-up, isto é, iniciando nos nós folhas e subindo até à raiz. Os valores de $A(v)$ e $B(v)$ são calculados como a seguir:

- Se v é um nó folha, $A(v) := c(v)$ e $B(v) := 0$.
- Se v não é um nó folha e tem filhos $x_1, ..., x_j$, $A(v) := max\{c(v) + B(x_1) + ... + B(x_j), A(x_1) + ... + A(x_j)\}$ e $B(v) := A(x_1) + ... + A(x_j)$.
  - $c(v) + B(x_1) + ... + B(x_j)$ - nesse caso, v é incluído no conjunto independente, e portanto, seus filhos não são;
  - $A(x_1) + ...A(x_j)$ - nesse caso, v não é incluído no conjunto independente e, portanto, seus filhos são incluídos;

O algoritmo construído dessa forma executa em tempo $O(|V|)$, pois cada valor $A(v)$ e $B(v)$ é calculado uma vez e usado somente uma vez (no momento em que o $A(v)$ e $B(v)$ de seu pai estão sendo calculados).

Na Figura 4.2, veja a árvore para a qual construímos uma exemplo do algoritmo.

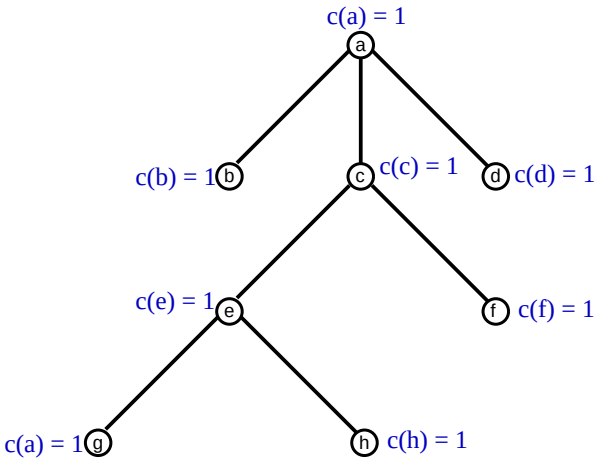

Figura 4.2: Exemplo da construção de solução do PCI para uma árvore

A solução é dada pelos valores a seguir:

Vértice	$A(v)$ e $B(v)$
g	$A(g) = 1$ $B(g) = 0$
h	$A(h) = 1$ $B(h) = 0$
f	$A(f) = 1$ $B(f) = 0$
b	$A(b) = 1$ $B(b) = 0$
d	$A(d) = 1$ $B(d) = 0$
e	$A(e) = max\{c(e) + B(g) + B(h), A(g) + A(h)\} = 2$ $B(e) = A(g) + A(h) = 2$
c	$A(c) = max\{c(c) + B(e) + B(f), A(f) + A(e)\} = 3$ $B(c) = A(e) + A(f) = 3$
a	$A(a) = max\{c(a) + B(b) + B(c) + B(d), A(b) + A(c) + A(d)\} = 5$ $B(a) = A(b) + A(c) + A(d) = 5$

4.3 Problema do conjunto independente ponderado em DEA

Nesta seção, apresentamos e exemplificamos então, uma forma de construir uma solução de PCI para grafos em geral utilizando a DEA do grafo. A técnica utilizada é chamada *computação de tabelas de caracterizações de soluções parciais*. Esta técnica é uma abordagem de programação dinâmica, inicialmente introduzida por Arnborg e Proskurowski [3], que a utilizaram para resolver problemas NP-Difíceis. Para simplificar, assuma que a DEA é Boa (veja como construí-la no Lema 3.10, página 41).

Considere a Boa-DEA $\Big(\{X_i | i \in I\}, T = (I, F)\Big)$ do grafo $G = \big(V, E\big)$. A cada nó $i \in I$ é associado um subgrafo $G_i = \big(V_i, E_i\big)$, sendo V_i a união de todas as bolsas X_j ($j = i$ ou j descendente de i em T) e E_i o conjunto de arestas em E que posuem ambos os extremos em V_i.

Para cada nó i é calculada uma tabela C_i. Cada tabela C_i conterá o peso máximo do conjunto independente para cada subconjunto $S \subseteq X_i$. Assim, considerando que a largura da DEA é k, cada tabela conterá no máximo 2^{k+1} valores.

Da mesma forma que na Sessão 4.2, as tabelas são calculadas de maneira bottom-up, isto é, das folhas para a raiz. Sendo assim, para resolver o problema original, devemos analisar a resposta que encontra-se na raiz da DEA. Assim podemos iniciar o algoritmo.

Se existirem dois vértices adjacentes em S, então faça $C_i(S) = -\infty$.

4.3.1 Construindo a tabela para um nó folha

Se i é um nó folha de T, por definição $|X_i| = 1$. Façamos $X_i = \{v\}$. A tabela, portanto, tem somente duas entradas e a tabela C_i é calculada como a seguir:

- $C_i(\emptyset) = 0$ e
- $C_i\big(\{v\}\big) = c(v)$

4.3.2 Construindo a tabela para um nó introduz

Se i é um nó introduz de T, então ele possui um único filho j e $X_i = X_j \cup \{v\}$. Em particular, temos que v não é adjacente a qualquer vértice em $V_i - X_j$.

Lema 4.1 Nó Introduz - *Seja* $S \subseteq X_j$

1. $C_i(S) = C_j(S)$
2. *Se existe um vértice* $w \in S$, *com* $(v, w) \in E$, *então* $C_i\big(S \cup \{v\} = -\infty\big)$
3. *Se para todo* $w \in S$, $(v, w) \notin E$, *então* $C_i\big(S \cup \{v\}\big) = C_j(S) + c(v)$

Prova.

1. Segue diretamente da observação de que para cada $S \subseteq X_i$, temos a coleção de conjuntos independentes W em G_i com $W \cap X_i = S$ sendo a mesma coleção de conjuntos independentes W em G_j com $W \cap X_j = S$.
2. Se $W \cap X_i = S \cup \{v\}$, então $\{v, w\} \subseteq W$ e, portanto, W não é um conjunto independente.
3. Seja W um conjunto independente em G_j com peso $C_j(S)$ e $W \cup X_i = S$, e suponha que para todo $w \in S : \{v, w\} \notin E$. Como todos os vizinhos de v em G_i devem pertencer a X_i, $W \cup \{v\}$ é um conjunto independente. Claramente, $\big(W \cup \{v\} \cap X_i = S \cup \{v\}\big)$, então $C_i\big(S \cup \{v\} \leq c\big(W \cup \{v\}\big)\big) = C_j(S) + c(v)$.

Seja W' um conjunto independente em G_i com $W' \cap X_i = S \cup \{v\}$ e peso $C_i\big(S \cup \{v\}\big)$. $W' - \{v\}$ é um conjunto independente em G_j com peso $C_j\big(S \cup \{v\} - c(v)\big)$ e $\big(W' - \{v\} \cap X_j = S\big)$. Então $C_j(S) \leq C_i\big(S \cup \{v\}\big) - c(v)$. ■

4.3.3 Construindo a tabela para um nó esquece

Se i é um nó esquece de T , então ele possui um único filho j e $X_i = X_j - \{v\}$.

Lema 4.2 Nó esquece - *Seja* $S \subseteq X_j$, $C_i(S) = max\{C_j(S), C_j\big(S \cup \{v\}\big)\}$.

Prova. Observe que $C_i(S)$ é o maior peso entre todos os conjuntos independentes W em G_i com $W \cap X_i = S$ e $v \in W$ e entre todos os conjuntos independentes W em G_i com $W \cap X_i = S$ e $v \notin W$. Como $G_i = G_j$, o primeiro termo se iguala a $C_j(S)$, e o segundo termo se iguala a $C_j\big(S \cup \{v\}\big)$. ■

4.3.4 Construindo a tabela para um nó junção

Se i é um nó junção de T, então ele possui dois filhos j_1 e j_2 e $X_i = X_{j_1} = X_{j_2}$. Podemos enxergar G_i como sendo um tipo de união de G_{j_1} e G_{j_2}. Assim, um conjunto independente em G_i tem uma parte em G_{j_1} e uma parte em G_{j_2} e no cálculo final temos que subtrair um termo, para nos certificarmos que não contabilizamos duas vezes um vértice que aparece em ambas as partes.

Lema 4.3 *Sejam i, j_1 e j_2 nós tal que $X_i = X_{j_1} = X_{j_2}$. Se $v \in X_{j_1}$, $w \in X_{j_2}$ e $v, w \notin X_i$, então $\{v, w\} \notin E$.*

Prova. É essencial o uso das propriedades de DEA. Suponha que $(v, w) \in E$. Então deve existir uma bolsa $X_{i'}$ com $\{v, w\} \in X_{i'}$. Suponha, sem perda de generalidade, que i' não pertença a subárvore de T_{j_1} , enraizada por j_1. (O caso em que i' não pertence à subárvore enraizada por j_2 é simétrico). Como $v \in V_{j_1}$, deve existir uma bolsa, $X_{i''}$, $i'' = j_1$ ou i'' um descendente de j_1 com $v \in X_{i''}$. Agora, como i está no caminho de i' até i'' em T, $v \in X_{i'}$ e temos que $v \in X_i$, o que é uma contradição. ■

Lema 4.4 Nó junção - *Seja $S \subseteq X_i$, $C_i(S) = C_{j_1}(S) + C_{j_2}(S) - c(S)$.*

Prova. Considere um conjunto independente W em G_i com peso $C_i(S)$ tal que $W \cap X_i = S$. $W \cap V_{j_1}$, é um conjunto independente em G_{j_1} com $W \cap X_{j_1} = W \cap X_i = S$; do mesmo modo para $W \subseteq X_{j_2}$ em G_{j_2}. Temos $W \cap V_{j_1} \cap V_{j_2} = W \cap X_i = S$. Agora

$$\begin{aligned} C_i(S) &= c(W) \\ &= c(W \cap V_{j_1}) + c(W \cap V_{j_2}) - c(W \cap V_{j_1} \cap V_{j_2}) \\ &= C_{j_1}(S) + C_{j_2}(S) - c(S) \end{aligned}$$

Suponha, que temos conjuntos independentes W_1 em G_{j_1} com peso $C_{j_1}(S)$ tal que $W_1 \cap X_{j_1} = S$, e W_2 com peso $C_{j_2}(S)$ tal que $W_2 \cap X_{j_2} = S$. Podemos afirmar que $W_1 \cup W_2$ é um conjunto independente em G_i . Considere dois vértices $\{v, w\} \in W_1 \cup W_2$. Como W_1 e W_2 são conjuntos independentes, v e w não são adjacentes quando ambos pertencem a W_1 ou a W_2. Então suponha, sem perda de generalidade que $v \in W_1$ e $w \in W_2$. Pelo Lema 4.3, se nem v e nem w pertencem a X_i, então v e w não são adjacentes. Se $v \in X_i$, então $v \in S$, e portanto $v \in W_2$, e v e w não são adjacentes como W é independente. Algo similar ocorre quando $w \in X_i$. E temos que

$$C_i(S) \geqslant c(W_1 \cup W_2)$$
$$= c(W_1) + c(W_2) - c(W_1 \cap W_2)$$
$$= C_{j_1}(S) + C_{j_2}(S) - c(S)$$

O resultado agora segue. ■

4.3.5 Construindo a solução com a utilização de todas as tabelas

Seja r a raiz de T. Depois de todas as outras tabelas terem sido calculadas, a tabela C_r é finalmente calculada. Dada essa tabela, é possível determinar o peso máximo de um conjunto independente em G.

Lema 4.5 *O peso máximo de um conjunto independente em G é $C_r(S)_{S \subseteq X_r}$.*

Prova. Note que $G_r = (V, E) = G$. O lema segue trivialmente. ■

A solução do PCIP da DEA da Figura 4.3 a seguir pode ser vista nas tabelas apresentadas posteriormente. Observe que são descritas as soluções C_i para $i = \{1, 2, ..., 23\}$.

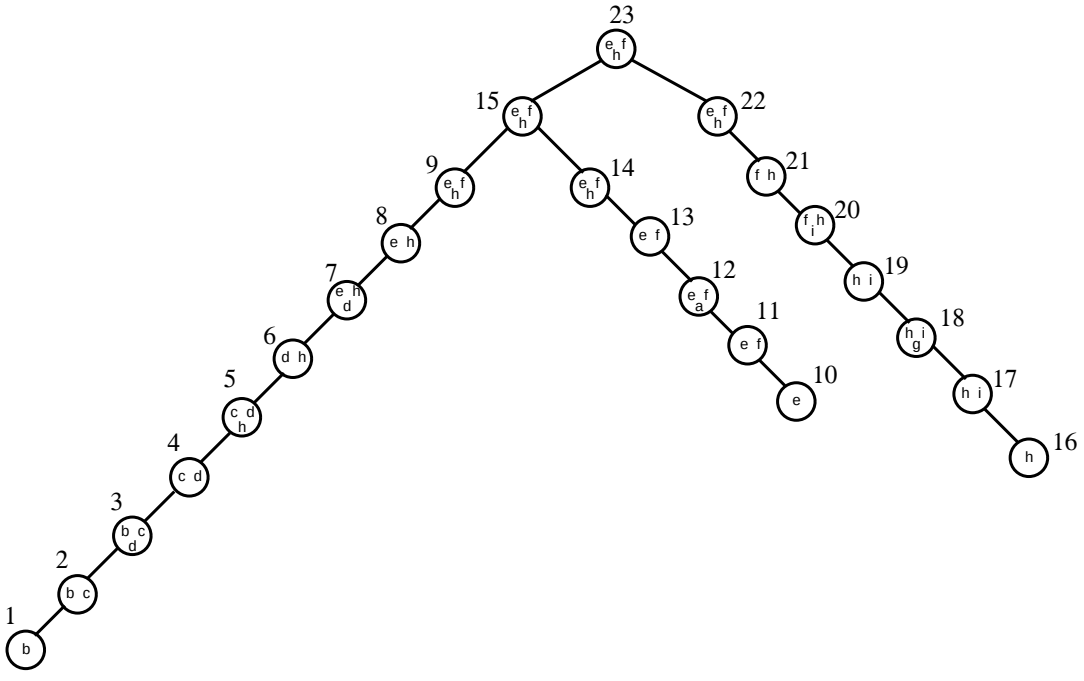

Figura 4.3: Boa-DEA e a numeração de seus nós

i	C_i
1	$C_1(\emptyset) = 0$
	$C_1(b) = c(b) = 1$
2	$C_2(\emptyset) = 0$
	$C_2(b) = C_1(b) = 1$
	$C_2(\emptyset \cup c) = 1$
	$C_2(b \cup c) = -\infty$
3	$C_3(\emptyset) = 0$
	$C_3(b) = C_2(b) = 1$
	$C_3(c) = C_2(c) = 1$
	$C_3(b \cup c) = C_2(b \cup c) = -\infty$
	$C_3(\emptyset \cup d) = C_2(\emptyset) + c(d) = 0 + 1 = 1$
	$C_3(b \cup d) = -\infty$
	$C_3(c \cup d) = -\infty$
	$C_3(b \cup c \cup d) = -\infty$
4	$C_4(\emptyset) = max\{C_3(\emptyset), C_3(\emptyset \cup b)\} = max\{0, 1\} = 1$
	$C_4(c) = max\{C_3(c), C_3(c \cup b)\} = max\{1, -\infty\} = 1$
	$C_4(d) = max\{C_3(d), C_3(d \cup b)\} = max\{1, -\infty\} = 1$
	$C_4(c \cup d) = max\{C_3(c \cup d), C_3((c \cup d) \cup b)\} = max\{-\infty, -\infty\} = -\infty$

i	C_i
5	$C_5(\emptyset) = C_4(\emptyset) = 1$
	$C_5(c) = C_4(c) = 1$
	$C_5(d) = C_4(d) = 1$
	$C_5(c \cup d) = -\infty$
	$C_5(\emptyset \cup h) = C_4(\emptyset) + c(h) = 0 + 1 = 1$
	$C_5(c \cup h) = -\infty$
	$C_5(d \cup h) = -\infty$
	$C_5((c \cup d) \cup h) = -\infty$
6	$C_6(\emptyset) = max\{C_5(\emptyset), C_5(\emptyset \cup c)\} = max\{1, 1\} = 1$
	$C_6(d) = max\{C_5(d), C_5(d \cup c)\} = max\{1, -\infty\} = 1$
	$C_6(h) = max\{C_5(h), C_5(h \cup c)\} = max\{1, -\infty\} = 1$
	$C_6(d \cup h) = max\{C_5(d \cup h), C_5((d \cup h) \cup c)\} = max\{-\infty, -\infty\} = -\infty$
7	$C_7(\emptyset) = C_6(\emptyset) = 1$
	$C_7(d) = C_6(d) = 1$
	$C_7(d \cup h) = -\infty$
	$C_7(\emptyset \cup e) = C_6(\emptyset) + c(e) = 1 + 1 = 2$
	$C_7(d \cup e) = -\infty$
	$C_7(h \cup e) = -\infty$
	$C_7((d \cup h) \cup e) = -\infty$
8	$C_8(\emptyset) = max\{C_7(\emptyset), C_7(\emptyset \cup d)\} = max\{1, 1\} = 1$
	$C_8(e) = max\{C_7(e), C_7(e \cup d)\} = max\{2, -\infty\} = 2$
	$C_8(h) = max\{C_7(h), C_7(h \cup d)\} = max\{1, -\infty\} = 1$
	$C_8(e \cup h) = max\{C_7(e \cup h), C_7((e \cup h) \cup d)\} = max\{-\infty, -\infty\} = -\infty$
9	$C_9(\emptyset) = C_8(\emptyset) = 1$
	$C_9(e) = C_8(e) = 2$
	$C_9(h) = C_8(h) = 1$
	$C_9(e \cup h) = -\infty$
	$C_9(\emptyset \cup f) = C_8(\emptyset \cup f) + c(f) = 1 + 1 = 2$
	$C_9(e \cup f) = -\infty$
	$C_9(h \cup f) = -\infty$
	$C_9((e \cup h) \cup f) = -\infty$
10	$C_{10}(\emptyset) = 0$
	$C_{10}(e) = c(e) = 1$

i	C_i
11	$C_{11}(\emptyset) = C_{10}(\emptyset) = 0$
	$C_{11}(e) = C_{10}(e) = 1$
	$C_{11}(\emptyset \cup f) = C_{10}(\emptyset) + c(f) = 1$
	$C_{11}(e \cup f) = -\infty$
12	$C_{12}(\emptyset) = C_{11}(\emptyset) = 0$
	$C_{12}(e) = C_{11}(e) = 1$
	$C_{12}(f) = C_{11}(f) = 1$
	$C_{12}(e \cup f) = -\infty$
	$C_{12}(\emptyset \cup a) = C_{11}(\emptyset \cup a) + c(a) = 0 + 1 = 1$
	$C_{11}(e \cup a) = -\infty$
	$C_{11}(f \cup a) = -\infty$
	$C_{11}((e \cup f) \cup a) = -\infty$
13	$C_{13}(\emptyset) = max\{C_{12}(\emptyset), C_{12}(\emptyset \cup a)\} = max\{0, 1\} = 1$
	$C_{13}(e) = max\{C_{12}(e), C_{12}(e \cup a)\} = max\{1, -\infty\} = 1$
	$C_{13}(f) = max\{C_{12}(f), C_{12}(f \cup a)\} = max\{1, -\infty\} = 1$
	$C_{13}(e \cup f) = max\{C_{12}(e \cup f), C_{12}((e \cup f) \cup a)\} = max\{-\infty, -\infty\} = 1$
14	$C_{14}(\emptyset) = C_{13}(\emptyset) = 1$
	$C_{14}(e) = C_{13}(e) = 1$
	$C_{14}(f) = C_{13}(f) = 1$
	$C_{14}(e \cup f) = -\infty$
	$C_{14}(\emptyset \cup h) = C_{13}(\emptyset) + c(h) = 1 + 1 = 2$
	$C_{14}(e \cup h) = -\infty$
	$C_{14}(f \cup h) = -\infty$
15	$C_{15}(\emptyset) = C_9(\emptyset) + C_{14}(\emptyset) - c(\emptyset) = 1 + 1 - 0 = 2$
	$C_{15}(e) = C_9(e) + C_{14}(e) - c(e) = 2 + 1 - 1 = 2$
	$C_{15}(f) = C_9(f) + C_{14}(f) - c(f) = 2 + 1 - 1 = 2$
	$C_{15}(h) = C_9(h) + C_{14}(h) - c(h) = 1 + 2 - 1 = 2$
	$C_{15}(e \cup f) = C_9(e \cup f) + C_{14}(e \cup f) - c(e \cup f) = -\infty$
	$C_{15}(e \cup h) = C_9(e \cup h) + C_{14}(e \cup h) - c(e \cup h) = -\infty$
	$C_{15}(f \cup h) = C_9(f \cup h) + C_{14}(f \cup h) - c(f \cup h) = -\infty$
	$C_{15}(e \cup f \cup h) = C_9(e \cup f \cup h) + C_{14}(e \cup f \cup h) - c(e \cup f \cup h) = -\infty$
16	$C_{16}(\emptyset) = c(\emptyset) = 0$
	$C_{16}(h) = c(h) = 1$

i	C_i
17	$C_{17}(\emptyset) = C_{16}(\emptyset) = 0$
	$C_{17}(h) = C_{16}(h) = 1$
	$C_{17}(\emptyset \cup i) = C_{16}(\emptyset) + c(i) = 0 + 1 = 1$
	$C_{17}(h \cup i) = -\infty$
18	$C_{18}(\emptyset) = C_{17}(\emptyset) = 0$
	$C_{18}(h) = C_{17}(h) = 1$
	$C_{18}(\emptyset \cup i) = C_{17}(\emptyset \cup i) = 1$
	$C_{18}(h \cup i) = C_{17}(h \cup i) = -\infty$
	$C_{18}(\emptyset \cup g) = C_{17}(\emptyset) + c(g) = 0 + 1 = 1$
	$C_{18}(h \cup g) = -\infty$
	$C_{18}(i \cup g) = -\infty$
	$C_{18}((h \cup i) \cup g) = -\infty$
19	$C_{19}(\emptyset) = max\{C_{18}(\emptyset), C_{18}(\emptyset \cup g)\}, max\{0, 1\} = 1$
	$C_{19}(h) = max\{C_{18}(h), C_{18}(h \cup g)\}, max\{1, -\infty\} = 1$
	$C_{19}(i) = max\{C_{18}(i), C_{18}(i \cup g)\}, max\{1, -\infty\} = 1$
	$C_{19}(h \cup i) = max\{C_{18}(h \cup i), C_{18}((h \cup i) \cup g)\}, max\{-\infty, -\infty\} = -\infty$
20	$C_{20}(\emptyset) = C_{19}(\emptyset) = 1$
	$C_{20}(h) = C_{19}(h) = 1$
	$C_{20}(i) = C_{19}(i) = 1$
	$C_{20}(h \cup i) = -\infty$
	$C_{20}(\emptyset \cup f) = C_{19}(\emptyset) + c(f) = 1 + 1 = 2$
	$C_{20}(h \cup f) = -\infty$
	$C_{20}(i \cup f) = -\infty$
	$C_{20}((h \cup i) \cup f) = -\infty$
21	$C_{21}(\emptyset) = max\{C_{20}(\emptyset), C_{20}(\emptyset \cup i)\} = max\{1, 1\} = 1$
	$C_{21}(f) = max\{C_{20}(f), C_{20}(f \cup i)\} = max\{2, -\infty\} = 2$
	$C_{21}(h) = max\{C_{20}(h), C_{20}(h \cup i)\} = max\{1, -\infty\} = 1$
	$C_{21}(f \cup h) = max\{C_{20}(f \cup h), C_{20}((f \cup h) \cup i)\} = max\{-\infty, -\infty\} = 1$
22	$C_{22} = (\emptyset) = C_{21}(\emptyset) = 1$
	$C_{22} = (f) = C_{21}(f) = 2$
	$C_{22} = (h) = C_{21}(h) = 1$
	$C_{22} = (f \cup h) = C_{21}(f \cup h) = -\infty$
	$C_{22} = (\emptyset \cup e) = C_{21}(\emptyset) + c(e) = 1 + 1 = 2$
	$C_{22} = (f \cup e) = -\infty$
	$C_{22} = (h \cup e) = -\infty$
	$C_{22} = ((f \cup h) \cup e) = -\infty$

i	C_i
23	$C_{23}(\emptyset) = C_{15}(\emptyset) + C_{22}(\emptyset) - c(\emptyset) = 2 + 1 - 0 = 3$
	$C_{23}(e) = C_{15}(e) + C_{22}(e) - c(e) = 2 + 2 - 1 = 3$
	$C_{23}(f) = C_{15}(f) + C_{22}(f) - c(f) = 2 + 2 - 1 = 3$
	$C_{23}(h) = C_{15}(h) + C_{22}(h) - c(h) = 2 + 1 - 1 = 2$
	$C_{23}(e \cup f) = C_{15}(e \cup f) + C_{22}(e \cup f) - c(e \cup f) = -\infty$
	$C_{23}(e \cup h) = C_{15}(e \cup h) + C_{22}(e \cup h) - c(e \cup h) = -\infty$
	$C_{23}(f \cup h) = C_{15}(f \cup h) + C_{22}(f \cup h) - c(f \cup h) = -\infty$
	$C_{23}(e \cup f \cup h) = C_{15}(e \cup f \cup h) + C_{22}(e \cup f \cup h) - c(e \cup f \cup h) = -\infty$

Na próxima seção, mostramos como resolver o problema de coloração por lista para grafos cordais utilizando a técnica de decomposição em árvore.

4.4 Problema de coloração por lista

Dado um grafo $G = \big(V, E\big)$ e um conjunto de cores $L(v)$, $\forall v \in V$, chamada *lista*, uma *coloração por lista* é uma função que mapeia todo vértice v a uma cor na lista $L(v)$. Como no problema de coloração, a coloração por lista deve ser *própria*, isto é, vértices adjacentes não possuem a mesma cor. Um grafo é *k-colorível* (ou *k-colorível por lista*) se ele tem uma coloração própria por lista. Assim, o problema consiste em decidir se G é k-colorível, isto é, possui uma k-coloração por lista.

Nesta seção reproduzimos a solução do problema da k-coloração por lista apresentada em [17]. Como contribuição, apresentamos um exemplo detalhado, onde é dado um grafo G com listas e, a partir de sua decomposição em árvore, verificamos se G é 3-colorível.

4.4.1 Coloração com listas

Na versão com listas de um problema de coloração, cada vértice do grafo de entrada tem uma lista das partes onde podem ser alocados.

Uma instância do problema *coloração com lista* é um grafo G, junto com uma coleção de listas $L(v)$, $\forall v \in V$, cada lista sendo um conjunto de partes. Uma solução para a instância da k-coloração com listas é uma solução para a k-coloração correspondente, tal que cada vértice v pertence a uma parte A_i, $i \in L(v)$.

A seguir, mostraremos como resolver o problema de coloração por lista, restrito a grafos cordais, utilizando a técnica de DEA.

4.4.2 Construindo a solução

Teorema 4.1 *[17] O problema da k-coloração com listas em grafos cordais pode ser resolvido em tempo polinomial.*

Prova. Um grafo cordal G que admite uma k-coloração com listas não pode conter uma clique com $k+1$ vértices; portanto deve possuir largura no máximo $k-1$. A existência de uma k-coloração com listas de um grafo de largura limitada pode ser testada por técnicas de programação dinâmica padrão [8, 15, 3].

Dada uma boa-DEA $D = \big(\chi, T\big)$ de G com raiz r, denotamos por G_i o subgrafo de G induzido pela união de X_i e todos os X_j onde j é um descendente de i. Seja $F(i)$ o conjunto de todos os pares $\big(\pi, S\big)$, onde π é uma atribuição dos vértices em X_i às partes, obtida restringindo uma k-coloração com listas Σ a G_i, e S é o conjunto daquelas partes na partição Σ que contém vértices de $G_i - X_i$. Observe que $F(i)$ tem no máximo $k^k 2^k$ elementos.

Podemos construir o conjunto $F(i)$ para cada nó, uma vez que os valores $F(j)$ de seus descendentes j já tenham sido calculados. Isso não é difícil de ver. Considere separadamente as folhas, os nós introduz, esquece e junção.

Se i é um nó folha, para cada $\big(\pi, S\big) \in F(i)$ consideramos todos os possíveis valores que v pode assumir em π restrito a sua lista de adjacência, e fazemos S vazio.

Se i é um nó esquece com um único filho j e $X_i = X_j - \{v\}$. Para cada $\big(\pi, S\big) \in F(j)$ adicionamos a $F(i)$ o par $\big(\pi', S'\big)$, onde π' corresponde a π restrito a X_i e S' é igual a S, se a parte A a qual v foi atribuído em π estiver presente em S, ou igual a $S \cup A$, caso contrário.

Se i é um nó introduz com um único filho j e $X_i = X_j + \{v\}$, então para cada $\big(\pi, S\big) \in F(j)$ consideramos todos os possíveis valores que v pode assumir com a atribuição π atual, por causa das adjacências de v em X_j, e também devido às não adjacências de v em $G_i - X_i$.

Se i é um nó junção com dois filhos X_{j_1} e X_{j_2}, tal que $X_i = X_{j_1} = X_{j_2}$, então para cada $\big(\pi_1, S_1\big) \in F(j_1)$ e $\big(\pi_2, S_2\big) \in F(j_2)$, faça a união das soluções tal que as adjacências e não adjacências sejam consideradas.

■

A seguir veja um exemplo de solução do problema de coloração por lista, restrito a grafos cordais, utilizando a boa-DEA da Figura 4.3. Veja na Figura 4.4 o grafo a ser colorido e as listas $L(v)$ restringindo as cores de cada vértice. Considere que cada elemento de $L(v)$ é uma cor possível.

i	F_i
1	$(\{b\},\{\emptyset\}) \Rightarrow (\{1\},\{\emptyset\})$
	$(\{b\},\{\emptyset\}) \Rightarrow (\{2\},\{\emptyset\})$
2	$(\{b,c\},\{\emptyset\}) \Rightarrow (\{1,2\},\{\emptyset\})$
	$(\{b,c\},\{\emptyset\}) \Rightarrow (\{1,3\},\{\emptyset\})$
	$(\{b,c\},\{\emptyset\}) \Rightarrow (\{2,3\},\{\emptyset\})$
3	$(\{b,c,d\},\{\emptyset\}) \Rightarrow (\{1,2,3\},\{\emptyset\})$
	$(\{b,c,d\},\{\emptyset\}) \Rightarrow (\{2,3,1\},\{\emptyset\})$
4	$(\{c,d\},\{b\}) \Rightarrow (\{2,3\},\{1\})$
	$(\{c,d\},\{b\}) \Rightarrow (\{3,1\},\{2\})$
5	$(\{c,d,h\},\{b\}) \Rightarrow (\{2,3,1\},\{1\})$
6	$(\{d,h\},\{b,c\}) \Rightarrow (\{3,1\},\{1,2\})$
7	$(\{d,h,e\},\{b,c\}) \Rightarrow (\{3,1,2\},\{1,2\})$
8	$(\{h,e\},\{b,c,d\}) \Rightarrow (\{1,2\},\{1,2,3\})$
9	$(\{h,e,f\},\{b,c,d\}) \Rightarrow (\{1,2,3\},\{1,2,3\})$
10	$(\{e\},\{\emptyset\}) \Rightarrow (\{2\},\{\emptyset\})$
	$(\{e\},\{\emptyset\}) \Rightarrow (\{3\},\{\emptyset\})$
11	$(\{e,f\},\{\emptyset\}) \Rightarrow (\{2,3\},\{\emptyset\})$
	$(\{e,f\},\{\emptyset\}) \Rightarrow (\{3,2\},\{\emptyset\})$
12	$(\{e,f,a\},\{\emptyset\}) \Rightarrow (\{2,3,1\},\{\emptyset\})$
	$(\{e,f,a\},\{\emptyset\}) \Rightarrow (\{3,2,1\},\{\emptyset\})$
13	$(\{e,f\},\{a\}) \Rightarrow (\{2,3\},\{1\})$
	$(\{e,f\},\{a\}) \Rightarrow (\{3,2\},\{1\})$
14	$(\{h,e,f\},\{a\}) \Rightarrow (\{1,2,3\},\{1\})$
	$(\{h,e,f\},\{a\}) \Rightarrow (\{1,3,2\},\{1\})$
15	$(\{h,e,f\},\{\{a,b\},\{c\},\{d\}) \Rightarrow (\{1,2,3\},\{1,2,3\})$
	$(\{h,e,f\},\{\{a,b\},\{c\},\{d\}) \Rightarrow (\{1,3,2\},\{1,2,3\})$
16	$(\{h\},\{\emptyset\}) \Rightarrow (\{1\},\{\emptyset\})$
17	$(\{h,i\},\{\emptyset\}) \Rightarrow (\{1,2\},\{\emptyset\})$
18	$(\{h,i,g\},\{\emptyset\}) \Rightarrow (\{1,2,3\},\{\emptyset\})$
19	$(\{h,i\},\{g\}) \Rightarrow (\{1,2\},\{3\})$
20	$(\{h,i,f\},\{g\}) \Rightarrow (\{1,2,3\},\{3\})$
21	$(\{h,f\},\{g,i\}) \Rightarrow (\{1,3\},\{3,2\})$
22	$(\{h,e,f\},\{g,i\}) \Rightarrow (\{1,2,3\},\{3,2\})$
23	$(\{h,e,f\},\{\{a,b\},\{c,i\},\{d,g\}\}) \Rightarrow (\{1,2,3\},\{1,2,3\})$

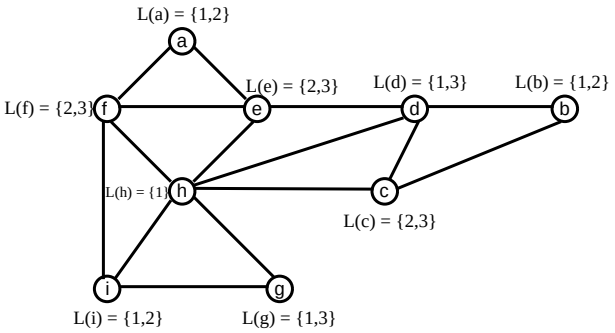

Figura 4.4: Grafo a ser colorido e listas de restrição

Observe que em $F(23)$ a solução

$$\big(\{h, e, f\}, \{\{a, b\}, \{c, i\}, \{d, g\}\}\big) \Rightarrow \big(\{1, 3, 2\}, \{1, 2, 3\}\big)$$

não é possível devido à aresta (f, i). Utilizando essa solução, os vértices f e i seriam, os dois, coloridos com a cor 2. Assim, o resultado do grafo G é dado pelo resultado obtido em $F(23)$.

Capítulo 5

Conclusão

A proposta deste trabalho foi fazer uma pesquisa aprofundada de uma técnica muito conhecida mundialmente, mas pouco estudada no Brasil - a técnica de decomposição em árvore (treewidth). O trabalho foi construído de forma que sirva de referência para o estudo da técnica, seja no entendimento das suas principais propriedades, seja na construção de decomposição em árvore para classes específicas de grafos, seja, no uso da técnica na resolução de problemas combinatoriais. Para isso, construímos um texto didático, sempre utilizando o auxílio das figuras, e fazendo com que todos os capítulos sejam interligados, mostrando desde as principais características até a utilização da técnica.

Acreditamos, analisando os capítulos, que os objetivos foram satisfatoriamente alcançados. No Capítulo 2 trabalhamos especificamente com as características das decomposições em árvores e os diferentes tipos de decomposições. No Capítulo 3 trabalhamos com a característica que faz com que algumas decomposições sejam melhores do que outras - a largura em árvore. Mostramos que diferentes decomposições em árvore podem ter diferentes larguras, e a melhor decomposição é aquela que possui menor largura. Nesse mesmo capítulo, ainda mostramos como construir uma decomposição em árvore ótima (com a menor largura possível) para a classe dos grafos cordais e para a classe das k-árvores. E, finalmente, no Capítulo 4 mostramos como resolver dois problemas combinatoriais utilizando a técnica. Ainda, no apêndice, mostramos que determinar a largura em árvore de um grafo é NP-Completo e um algoritmo linear que testa se um dado grafo G tem largura em árvore no máximo k, k constante, e, em caso afirmativo, retorna uma decomposição em árvore de G com largura em árvore no máximo k.

5.1 Trabalhos Futuros

Como trabalhos futuros, sugerimos os seguintes problemas:

- Estudar outras técnicas de triangularizações, já que estas fornecem um limite superior para a largura em árvore do grafo.
- Estudar classes de grafos que possuem largura em árvore limitada.
- Estudar algoritmos para construção de decomposição em árvore para várias classes de grafos.
- Estudar outros problemas combinatoriais que podem ser resolvidos utilizando a técnica de decomposição em árvore.

APÊNDICE A – Complexidade em encontrar decomposições em árvore

A.1 Introdução

Como vimos no Capítulo 4 é extremamente útil transformar um grafo de qualquer outra classe em uma árvore. Mas neste momento surge uma questão: qual é o custo em se transformar um grafo de qualquer classe em uma árvore? Isto é, qual a complexidade em determinar a largura em árvore de certas classes de grafos?

O problema largura em árvore k (dado um grafo G, G tem largura em árvore no máximo k?) pode ser resolvido em tempo linear [6]. Nesse caso, k é uma constante e não parte da entrada. Se a resposta é sim, o algoritmo de Bodlaender também constrói uma DEA ótima em tempo linear. Assim, para grafos que têm largura em árvore limitada por uma constante, sua largura em árvore e sua DEA ótima podem ser construídas em tempo linear. Isso é importante considerando que é necessário uma DEA ótima quando projetamos algoritmos de tempo polinomial para grafos de largura em árvore limitada.

Infelizmente, o problema largura em árvore (dado um grafo G e um inteiro $k < |V|$, G tem largura em árvore no máximo k?) é um problema NP-Difícil [1]. Nesse caso, k faz parte da entrada.

Neste capítulo nós estamos justamente interessados em determinar a complexidade dos problemas acima. Começamos provando que o problema largura em árvore k é polinomial. E seguimos com a prova de que o problema largura em árvore é NP-Completo.

A.2 Algoritmo de tempo linear para o problema largura em árvore k

Nesta seção apresentamos o algoritmo de tempo linear para a solução do problema largura em árvore k [6, 18], proposto por Bodlaender em [6] . Esse resultado é resumido pelo seguinte Teorema:

Teorema A.1 *Para todo $k \in \mathbb{N}$, existe um algoritmo de tempo linear que testa se um dado grafo $G = (V, E)$ tem largura em árvore no máximo k, e, em caso positivo, retorna uma DEA de G com largura em árvore no máximo k.*

O algoritmo é dividido em duas partes. A idéia da parte 1 do algoritmo se resume em, a partir do grafo G, recursivamente, contrair algumas arestas "especias" formando um novo grafo G' e construindo, se possível, uma DEA de G' de largura no máximo k. Tendo essa decomposição, é possível construir uma DEA de G de largura no máximo $2k + 1$ e, então, finalmente, encontrar uma DEA de G de largura k. Na parte 2, recursivamente, um grafo G' é formado a partir da adição de algumas arestas e remoção de alguns vértices com características específicas e é construída uma DEA de G' de largura no máximo k. Dada tal decomposição é, então, possível construir uma DEA de G de largura no máximo k.

Na próxima seção são apresentadas algumas definições necessárias para o entendimento do algoritmo, assim como os lemas e teoremas que dão embasamento ao funcionamento do mesmo.

A.2.1 Preliminares

Na Parte 1 do algoritmo uma DEA D' com largura $2k+1$ é construída para G. Através do algoritmo apresentado no seguinte Teorema, uma DEA D com largura k é construída para G, se possível.

Teorema A.2 *[7, 18] Para todo k, l, existe um algoritmo de tempo linear que, dados um grafo $G = (V, E)$ e uma decomposição em árvore (χ, T) de G com largura no máximo l, determina se a largura em árvore de G é no máximo k, e, em caso positivo, encontra uma decomposição em árvore de G com largura no máximo k.*

Nesta seção, chamaremos de *DEA-Clique* a DEA $D = (\chi, T = (I, F))$ para um grafo G tal que para todo $i \in I$, X_i é uma clique maximal em G e tal que para todo $i, j \in I$,

se $i \neq j$, então $X_i \neq X_j$. Vimos no Teorema 3.1 que todo grafo cordal possui tal decomposição em árvore. O Lema A.1 a seguir nós fornece um limite para a quantidade de nós em uma DEA-Clique.

Lema A.1 *Se $(\chi, T = (I, F))$ é uma DEA-Clique de $G = (V, E)$ com largura k, então $|I| = |V| - k$.*

Prova. Por indução em $|I|$. Se $|I| = 1$, então claramente $|V| = k + 1$.

Suponha que o lema ocorra para para $|I| = r - 1$. Considere a DEA-Clique (χ, T) de G com largura k, com $|I| = r$. Seja f uma folha de T e seja j o único vizinho de f. Se $X_f \subseteq X_j$, então podemos remover f de T e continuarmos com a DEA restante. Se existe um único vértice v que pertence a X_f, mas não pertence a X_j e, portanto a qualquer outro conjunto X_i, $i \in I - \{j, f\}$, remova o nó f de T. Assim, obtemos uma DEA de $G[V - \{v\}]$ com largura k e $|I|$ nós. E o resultado segue por indução. ■

Seja $l \geq 1$ algum inteiro (que será determinado mais tarde).

Uma *coleção folha-caminho de T* é um conjunto de folhas e um conjunto de caminhos na DEA de tamanho l (isto é, com l nós), tal que nenhuma folha pertença a um caminho e quaisquer dois caminhos não possua um nó em comum.

Lema A.2 *T tem uma coleção folha-caminho C com pelo menos $\frac{n-k}{2l}$ elementos.*

Prova. Sejam f o número de folhas, d_2 o número de vértices de grau dois e d_3 o número de vértices de grau maior ou igual a três. Como cada nó de grau no mínimo três gera pelo menos três folhas, concluímos que $d_3 < f$. Todos os nós de grau dois pertencem a menos de $d_3 + f$ componentes conexas, obtidas pela remoção de todas as folhas e todos os nós com grau maior ou igual a três da árvore (com a retirada de um nó de grau pelo menos três, a quantidade de componentes formadas é no máximo a quantidade de folhas geradas pelo nó retirado). Cada componente contém pelo menos $l-1$ nós que não pertecem a nenhuma coleção folha-caminho de tamanho máximo. Então existem menos que $(d_3 + f)(l - 1)$ nós de grau dois que não estão em nenhum caminho da coleção. Logo, existem pelo menos

$$\frac{d_2-(d_3-f)(l-1)}{l}$$

caminhos em uma coleção folha-caminho de tamanho máximo. Segue que o tamanho máximo da coleção folha-caminho é pelo menos

$$max\Big(f, \frac{d_2-(d_3-f)(l-1)}{l} + f\Big) \geq \frac{1}{2}\frac{r}{l} = \frac{r}{2l}.$$

■

Um *emparelhamento* num grafo G é um conjunto M de arestas dotado da seguinte propriedade: todo vértice de G incide em no máximo um elemento de M. Um emparelhamento é *máximal* se não estiver contido em nenhum outro maior . Um emparelhamento maximal pode ser encontrado em tempo linear [11].

Seja $d \geq k+1$ uma constante (a ser determinada posteriormente).

Seja H o conjunto de *vértices de alto grau*, isto é, aqueles vértices com grau maior que d.

Da mesma forma, seja L o conjunto de *vértices de baixo grau*, isto é, aqueles vértices com grau no máximo d. Particionamos L em dois conjuntos L_I e L_M: L_M são aqueles vértices de L que tem pelos menos um vizinho em L; L_I são aqueles vértices de L sem qualquer vizinho em L, ou seja, podem ser adjacentes somente aos vértices de alto grau em H. Observe que os vértices de L_I são isolados no grafo induzido pelos vértices de L ($G[L]$).

Observe que as quantidades de vértices de alto e baixo grau que determinam o que o algoritmo deve fazer - i.e., se ele dever adicionar ou remover arestas. O próximo Lema determina a quantidade máxima de vértices de alto grau que um grafo deve possuir para que tenha largura em árvore no máximo k.

Lema A.3 *Se $G = \big(V, E\big)$ tem largura em árvore no máximo k, então $|H| \leq \frac{2|V|k}{d}$.*

Prova. O número de arestas em G é pelo menos $|E| \geq \frac{|H|.d}{2}$. Pelo Lema 3.8, o número de arestas é $|E| \leq k.|V| - \frac{1}{2}k(k+1) \Rightarrow |E| \leq k.|V|$. Assim,

$$k.|V| \geq \frac{|H|.d}{2} \; |H| \leq \frac{2k|V|}{d}$$

■

Na primeira parte do algoritmo, as arestas do emparelhamento maximal são contraídas formando um novo grafo, e a partir deste, verifica-se a possibilidade de construir uma DEA de largura no máximo k para o grafo original G. Assim, é necessário um limite mínimo no

número de arestas para que seja interessante fazer tal contração. O Lema a seguir limita esse número com base na quantidade de vértices do conjunto L_M.

Lema A.4 *Todo emparelhamento maximal de G tem pelo menos $\frac{|L_M|}{2d}$ arestas.*

Prova. Considere um emparelhamento maximal M. Qualquer vértice de L_M deve ser extremo de uma aresta em M. Para cada aresta e de M, podemos associar no máximo $2d$ vértices de L_M que são extremos de e ou adjacentes a um vértice de L_M. Então $|M| \geq \frac{|L_M|}{2d}$. ■

Seja P uma coleção folha-caminho de uma DEA-clique $D = (\chi, T = (I, F))$ com caminhos de tamanho l, com pelo menos $\frac{n-k}{l}$ elementos. Para um caminho $C \in P$, denote por $I(P)$ o seu conjunto de nós e por C_0 o interior, com nós $I(C_0)$. Para um conjunto de nós $J \subseteq I$ de nós, faça $X(J) = \bigcup_{i \in J} X_i$.

Seja $Z = (V, C)$ uma *imersão em uma k-árvore* de G, isto é, Z é subgrafo de uma k-árvore e G é subgrafo de G.

Um vértice $x \in L_I$ é chamado *pendurado em D* se existe um nó $i \in I$ tal que $N(x) \subseteq X_i$.

Lema A.5 *Se $d \geq k + l$ e $l \geq k^2 + 3k + 4$, então todo caminho em DEA $C \in P$ tem pelo menos um nó $i \in I(C_0)$ tal que X_i tem um vértice que é pendurado ou é um elemento de L_M e que não está em X_j para qualquer $j \notin I(C_0)$.*

Prova. Seja $C = (1, 2, ..., l)$ um caminho em P. Note que $|X(I(C))| = k + l \leq d$. Então todos os vértices de $X(I(C)) - (X_1 \cup X_l)$ são vértices de baixo grau. Seja Q o conjunto desses vértices, isto é, $Q = X_2 \cup X_3 \cup ... \cup X_{l-1}$. Se um desses vértices está em L_M e considerando que esse vértice não está nem em X_1 nem X_l, a prova está terminada.

Suponha agora que todos os vértices de Q são adjacentes somente a vértices de alto grau em $X_1 \cup X_l$ e que $H \cap X_1 = \{v_1, ..., v_r\}$. Claramente, $r \leq |X_1| = k + 1$. Assuma que v_i ocorre nos sucessivos subconjuntos $X_1, ..., X_{w_i}$. Suponha sem perda de generalidade que $w_1 \leq ... \leq w_r$.

Caso 1: Assuma que algum vértice $v \in Q$ ocorre somente em *um* único subconjunto X_j, para algum $2 \leq j \leq l - 1$. Nessa caso v é pendurado por definição.

Caso 2: Assuma que algum vértice $v \in Q$ ocorre somente em alguns subconjuntos $X_{w_i+1}, ..., X_{w_{i+1}}$ para algum i. Nesse caso, todos os vizinhos de x estão contidos em $X_{w_{i+1}}$. Assim v é pendurado.

Caso 3: Se nenhum dos casos anteriores ocorrer, todos os vértices de $X(I(C))$ devem pertencer a pelo menos um dos subconjuntos $X_1, X_{w_1}, ..., X_{w_r}, X_l$. Assim, existem no total $(k+1)(k+3) = k^2 + 4k + 3 = l - 1$ vértices onde $k+1$ é a quantidade de vértices de cada bolsa X_i e $k+3$ é a quantidade de bolsas - X_1, X_l e $X_{w_1}, ..., X_{w_r}$) que é menor que $k + l = |X(I(C))|$. Contradição.

■

Teorema A.3 *Se $d \geq k + l$ e $l \geq k^2 + 3k + 4$ então existem pelo menos $\frac{n-k}{2l}$ vértices que estão em L_M ou são pendurados.*

Prova. Considere uma coleção folha-caminho P com pelo menos $\frac{n-k}{2l}$ elementos (Lema A.2).

Se i é uma folha de T, então X_i contém um vértice x que é simplicial em Z. Então $N_G(x) \subseteq N_Z(x) \subset X_i$. Então esse vértice está em L_M ou é pendurado em D.

Pelo Lema A.5, todo caminho $C \in P$ tem um nó $i \in I(C_0)$ tal que X_i tem um vértice que está em L_M ou é pendurado em D. ■

O *grafo parcialmente preenchido* G' é o grafo obtido de G como a seguir: se dois vértices x e y de H tem pelo menos $k+1$ vizinhos em comum em L, então x e y são adjacentes em G'.

Lema A.6 *Suponha que v e w tem pelo menos $k+1$ vizinhos em comum em $G = (V, E)$. Se a largura de G é no máximo k, então a largura em árvore de $G + (v, w)$ é também no máximo k. Além disso, qualquer DEA de G com largura no máximo k é também uma DEA de $G + (v, w)$ com largura no máximo k e vice-versa.*

Prova. Suponha que (χ, T) é uma DEA de G com largura no máximo k. Pelo Lema 2.3(2) existem duas situações possíveis:

- Existe um $i \in I$ com $v, w \in X_i$, e nesse caso o lema segue diretamente.

- Existe um $i \in I$ com X_i contendo o conjunto W de todos os vizinhos de v e w. Agora, se adicionarmos arestas entre todos os vértices não adjacentes em W, obtemos um grafo G' que contém uma clique com pelo menos $k+2$ vértices e tem a DEA (χ, T). Mas um grafo com largura em árvore menor ou igual a k não pode conter uma clique com mais que $k+1$ vértices. Contradição. ■

Lema A.7 *Se a largura em árvore de G é no máximo k, então a largura em árvore do grafo preenchido de G é no máximo k. Além disso, qualquer DEA de G com largura no máximo k é tmbém uma DEA do grafo parcialmente preenchido com largura no máximo k, e vice-versa.*

Prova. Segue diretamente do Lema A.6. ■

Seja novamente $Z = (V, C)$ uma imersão em uma k-árvore de G (e, portanto, também do grafo parcialmente preenchido G'). Considere $D = (\chi, T = (I, F))$ uma DEA-clique de Z. Um subconjunto $Y \subseteq Z$ é um *conjunto-clique em D* se existe um nó $i \in I$ tal que $Y \subseteq X_i$. Um conjunto-clique é *maximal* se ele não está propriamente contido em outro conjunto-clique.

Lema A.8 *Existem no máximo $|H|$ conjuntos-cliques maximais.*

Prova. Considere o subgrafo induzido $Z[H]$. Cada conjunto clique maximal é um conjunto-clique maximal em $Z[H]$, pois estamos considerando que $d \geq k+1$. O número de cliques maximais em $Z[H]$ é limitado pelo número de vértices, já que esse grafo é cordal. ■

Teorema A.4 *Existem no máximo $\frac{1}{2}|H|k^2(k+1)$ vértices pendurados não simpliciais no grafo parcialmente preenchido G'.*

Prova. Considere os vértices pendurados em L_I que não são simpliciais em G'. Para cada um, escolha um par de vizinhos em H que não são adjacentes. Note que tal par de vizinhos está contido em algum conjunto-clique maximal.

Cada par de vértices não adjacentes pode estar associado a no máximo k vértices pois, caso contrário, pela definição de grafo parcialmente preenchido, eles deveriam ser

adjacentes em G'. Pelo Lema A.8, existem no máximo $|H|$ conjuntos-clique maximais, e cada um tem no máximo $\binom{k+1}{2}$ pares de vértices não adjacentes. Então existem no máximo $|H|k\binom{k+1}{2}$ vértices pendurados não simpliciais em G'. ■

Teorema A.5 *Existem constantes $\varepsilon = \varepsilon(k) > 0$ e $\delta = \delta(k) > 0$ tal que para todo grafo G com pelo menos $k+1$ vértices, pelo menos uma das seguintes propriedades ocorre.*

1. *A largura em árvore de G é maior que k*
2. *Todo emparelhamento maximal tem pelo menos εn arestas*
3. *Existem pelo menos δn vértices em L_I que são simpliciais no grafo parcialmente preenchido G'.*

Prova. Escolha $l \geq k^2 + 3k + 4$ e $d \geq k + l$. Pelo Teorema A.3 existem pelo menos $\frac{n-k}{2l} \geq \frac{n}{2l(k+1)}$ vértices em L_M ou pendurados.

Pelo Teorema A.4 e Lema A.3, dentre os vértices pendurados, existem no máximo $\frac{nk^3(k+1)}{d}$ vértices que não são simpliciais no grafo parcialmente preenchido. Assim, pelo menos

$$n\left(\frac{1}{2l(k+1)} - \frac{|L_M|}{n} - \frac{k^3(k+1)}{d}\right)$$

vértices são simpliciais no grafo parcialmente preenchido G'.

Escolha $d \geq 2lk^3(k+1)^2$,

$$0 < \varepsilon < \frac{1}{2d}\left(\frac{1}{2l(k+1)} - \frac{k^3(k+1)}{d}\right)$$
$$0 < \delta < \frac{1}{2l(k+1)} - 2\varepsilon d - \frac{k^3(k+1)}{d}$$

Então pelo Lema A.4, se $|L_M| \geq 2\varepsilon dn$, todo emparelhamento maximal tem pelo menos εn arestas. Por outro lado, se $|L_M| < 2\varepsilon dn$, pelo menos δn vértices são simpliciais no grafo parcialmente preenchido. ■

A.2.2 Algoritmo principal

O algoritmo completo está descrito a seguir.

Algoritmo A.1 *Algoritmo de tempo linear para DEA*

- *Algoritmo: DEA em tempo linear - Parte 1*

1. *Se $n \leq k+1$, a largura em árvore de G é no máximo k, então* **Pare**.
2. *Teste se $|E| \leq k.|V| - \frac{1}{2}k(k+1)$. Caso contrário, pelo Lema 3.8, G tem largura em árvore maior que k, então* **Pare**.
3. *Determine o emparelhamento maximal M.*
4. *Se $|M| \geq \varepsilon n$, então faça o seguinte: (Teorema A.5)*
   (a) *Contraia todas as arestas de M, obtendo o grafo $G(M)$.*
   (b) *Recursivamente aplique o algoritmo a $G(M)$.*
   (c) *Se $LA(G(M)) > k$, então $LA(G) > k$.* **Pare**.
   (d) *Se $LA(G(M)) \leq k$, o algoritmo constrói uma DEA D' para $G(M)$ com largura no máximo k. Modifique D' e construa uma DEA D para G com largura no máximo $2k+1$.*
   (e) *Use D e o algoritmo do Teorema A.2 para decidir se $LA(G) > k$ ou para construir a DEA de G com largura no máximo k.*

   - *Algoritmo: DEA em tempo linear - Parte 2*

5. *Assuma $|M| < \varepsilon n$. (Teorema A.5)*
   (a) *Construa o grafo parcialmente preenchido G'.*
   (b) *Encontre S o conjunto de todos os vértices em L_I que são simpliciais em G'. Se algum vértice $x \in S$ tem mais que k vizinhos, então $LA(G) > k$.* **Pare**.
   (c) *Se $|S| < \delta n$, então $LA(G) > k$.* **Pare**. *(Teorema A.5)*
   (d) *Se $|S| \geq \delta n$, construa $G(S) = G'[V - S]$.*
   (e) *Recursivamente, aplique o algoritmo a $G(S)$. Se $LA(G(S)) > k$, então $LA(G) > k$.* **Pare**.

(f) Caso contrário, o algoritmo retorna uma DEA D' de $G(S)$ com largura no máximo k. Use D' para obter uma DEA D para G com largura no máximo k.

A seguir mostramos que o algoritmo apresentado possui tempo linear de execução, considerando os passos não-recursivos e recursivos.

Na Parte 1 trabalhamos com emparelhamento. Em [11] Edmonds mostra que é possível determinar o emparelhamento maximal de um grafo G em tempo linear. Seja $M = \{(x_1, y_1), ..., (x_t, y_t)\}$. Seja $G(M)$ o grafo obtido de G pela contração das arestas de M. Então, se G tem n vértices, $G(M)$ tem $n - t$ vértices. Veja como construir $G(M)$ em tempo linear:

- Primeiro caminhe por todas as listas de adjacências de todos os vértices de G e substitua todas as ocorrências de cada y_i por x_i.
- Remova todos os vértices y_i, $(i = 1, ..., t)$ do grafo.
- Remova as arestas duplicadas. Isso pode ser feito linearmente através da ordenação das listas de adjacências.

Sendo $G(M)$ um menor de G, pela Propriedade 2.3 temos que a largura em árvore de $G(M)$ é no máximo igual a largura em árvore de G.

Agora, considere uma DEA $D' = (\chi', T')$ para $G(M)$ com largura no máximo k. É possível transformá-la em uma DEA $D = (\chi, T)$ para G como a seguir:

- Para cada $X_i \in \chi'$ contendo uma aresta contraída $e = (x, y)$. Substitua e por seus extremos em $X_i \in \chi$.

Obviamente, D construído como acima é uma DEA para G e possui largura no máximo $2k + 1$. Assim, sendo k uma constante, podemos usar D e D' e aplicar o algoritmo apresentando no Teorema A.2 para obter uma DEA para G com largura k.

Na Parte 2 trabalhamos com grafo parcialmente preenchido e seus vértices simpliciais.

Vejamos agora como construir um grafo parcialmente preenchido em tempo linear. Os passos são dados a seguir:

- Determine os conjuntos H, L_I e L_M.

- Para todo par de vértices $\{x, y\}$ em H com pelo menos $k+1$ vizinhos em L, adicione a aresta (x, y) - se ela ainda não estiver presente.
- Numere os vértices de H, digamos $v_1, ..., v_t$.
- Faça uma fila Q contendo entradas da forma $\big((v_i, v_j), -\big)$ ou $\big((v_i, v_j), x\big)$, onde $i < j$, v_i e v_j são vértices de H e x é um vizinho em comum de v_i e v_j em L.
  - Para toda aresta (v_i, v_j) com $i < j$ em H, coloque $\big((v_i, v_j), -\big)$ em Q.
  - Para todo vértice $x \in L$ e para todo par de vértices $\{x_i, y_i\}$ com $i < j$ em $N_G(x) \cap H$, coloque $\big((v_i, v_j), x\big)$ em Q.
- Use o algoritmo *bucket sort*, que possui complexidade $O(n)$, duas vezes para ordenar as entradas de Q em ordem alfabética os pares (v_i, v_j). Feito isso, todas as entradas da forma $((v_i, v_j), ...)$ ocorrem consecutivamente.
- Agora é fácil de contar para cada par $\{v_i, v_j\}$ o número de vizinhos em comum. Se esse número é maior que $k + 1$, e se a entrada $\big((v_i, v_j), -\big)$ não está presente em Q, adicionamos a aresta (v_i, v_j) no grafo.

Por fim, mostramos como determinar os vértices pertencentes a L_I que são simpliciais em G' em tempo linear - o conjunto S:

- Verifique a fila ordenada Q - para cada aresta (v_i, v_j) e para cada $x \in L_I$ tal que $\big((v_i, v_j), x\big) \in Q$ e tal que v_i $\big((v_i, v_j), -\big) \in Q$ ou v_i ev_j tem $k+1$ vizinhos em comum em L, adicione um ao contador de x.

Note que o algoritmo é recursivamente aplicado a $G(M)$. Como $|M| \geq \varepsilon n$, $G(M)$ tem no máximo $n - \varepsilon n = n(1 - \varepsilon)$ vértices.

Removendo todos os vértices de S do grafo G' obtemos o grafo $G(S)$. O algoritmo é recursivamente aplicado ao grafo $G(S)$ que tem no máximo $n - \delta n = n(1 - \delta)$ vértices. Claramente, se algum vértice de S tem mais que k vizinhos, a largura am árvore de G' é maior que k, e portanto a largura em árvore de G também.

Portanto, como ε e δ são positivos, o algoritmo precisa de tempo linear para executar os passos recursivos.

A.3 NP-Completude do problema largura em árvore

A.3.1 Definições

Um *grafo parcial (subgrafo gerador)* H de um grafo G contém todos os vértices de G e um subconjunto de suas arestas.

Um *supergrafo* H de um grafo G é qualquer grafo H do qual G é um grafo parcial.

Na Figura A.1 veja exemplos dessas três definições dadas.

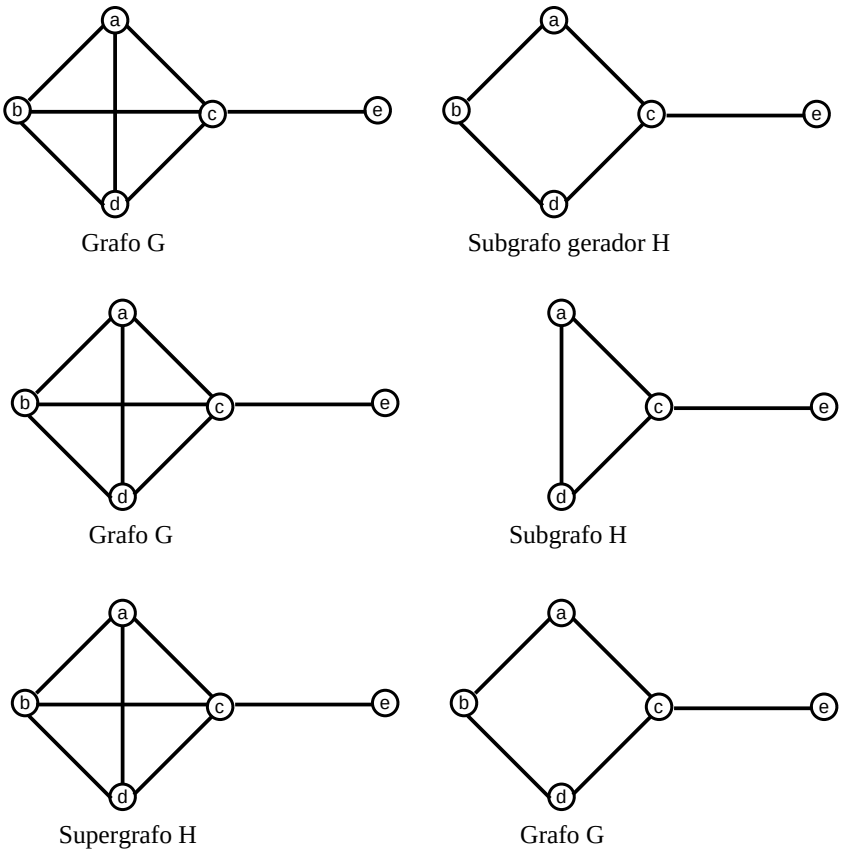

Figura A.1: Exemplos de grafo parcial (subgrafo gerador), subgrafo e supergrafo

A *vizinhança fechada* $V'(u)$ de um vértice $v \in V$ é o conjunto de todos os vértices adjacentes a v união ao próprio v, $V'(v) = \{u : (u, v) \in E\} \cup \{v\}$. Ou seja, é a vizinhança aberta contendo também o vértice v.

Um grafo G é *k-decomponível* se e somente se,

- G tem no máximo $k + 1$ vértices
- Ou existe um subgrafo S de G com no máximo k vértices tal que $G - S$ é desconexo,

e cada uma das suas componentes de $G - S$ acrescentadas a S (S completamente conexo) é k-decomponível.

Um grafo k-decomponível é apresentado na Figura A.2. Observe que no exemplo é apresentado um grafo 3-decomponível, e ao retirar o subgrafo S, o grafo é divido em duas componentes C_1 e C_2.

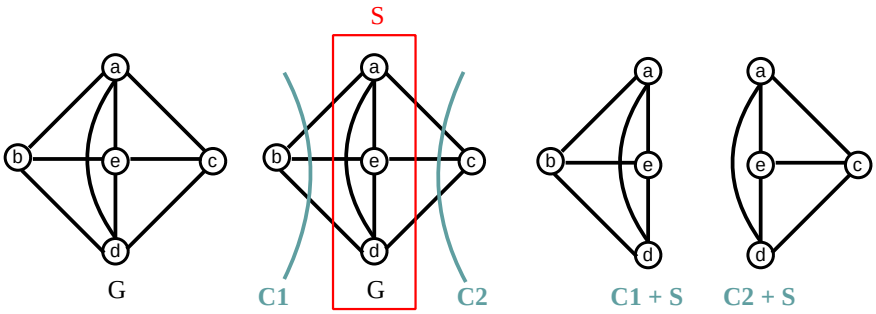

Figura A.2: Exemplo de um grafo k-decomponível

Um *esquema de eliminação* de um grafo é uma ordenação π de seus vértices.

Uma aresta (u, w) é uma aresta de preenchimento se existe um vértice v precedendo u e w no esquema de eliminação π tal que u e w são adjacentes a v, via aresta original ou via aresta de preenchimento, e u e w não são adjacentes entre si, mas tornam-se adjacentes pela adição de uma aresta (de preenchimento).

O conjunto completo de arestas de preenchimento é facilmente obtido verificando os vértices de π, e isso pode ser feito lineamente em $O(n)$.

O *grafo preenchido de* $G = (V, E)$ *com respeito à* π é o grafo $G(V, E \cup F)$ onde F são arestas de preenchimento. A Figura A.3 mostra dois exemplos de grafos de preenchimento com respeito a dois diferentes esquemas de eliminação.

Dado A, um subgrafo completo de G, dizemos que G é um caminho A-cordal se existe um eep π tal que se u imediatamente segue v em π, então u e v são adjacente e os vértices de A estão por último em π.

Um grafo G é um *grafo caminho-cordal* se, e somente se, existe um A para o qual G é um caminho A-cordal. Na Figura A.4 é mostrado um exemplo de um caminho A-cordal, que por conseqüência, é um grafo caminho-cordal.

Um *grafo k-cordal* é um grafo cordal G para o qual $\omega(G) = k + 1$. Observe que temos no exemplo da Figura A.4 um grafo 3-cordal, pois $\omega(G) = 3 + 1$.

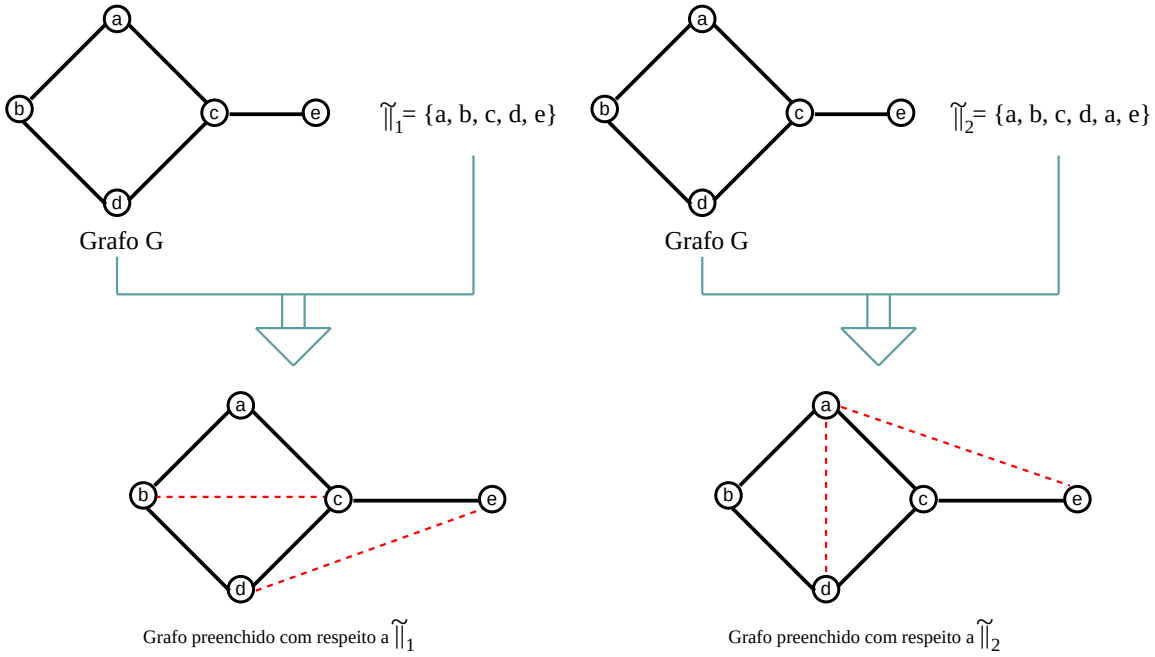

Figura A.3: Arestas de preenchimento e grafo preenchido com respeito a π

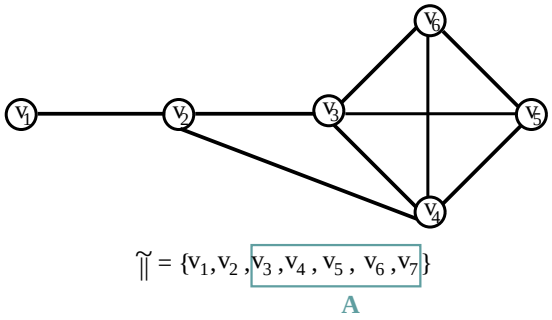

Figura A.4: Caminho A-cordal e grafo caminho-cordal

$k_t(G)$ é o mínimo k para o qual G é uma k-árvore parcial.

$k_c(G)$ é o mínimo k para o qual G é um grafo k-cordal.

Um *bloco* de um grafo G é um conjunto maximal de vértices com a mesma vizinhança fechada.

Claramente, os blocos de G particionam V.

Um *esquema de eliminação bloco-contínuo* é um esquema em que os vértices de cada bloco são eliminados seguidamente, isto é, um imediatamente após o outro.

Um grafo bipartido $G(A \cup B, E)$ é um *grafo cadeia* [32] se os vizinhos dos vértices em A formam uma cadeia, isto é, existe uma bijeção $\tau : A \leftrightarrow \{1, 2, ..., |A|\}$ tal que $\tau(u) < \tau(v)$ se, e somente se, $V(u) \supseteq V(v)$. Tal permutação τ é chamada *ordenação em cadeia*.

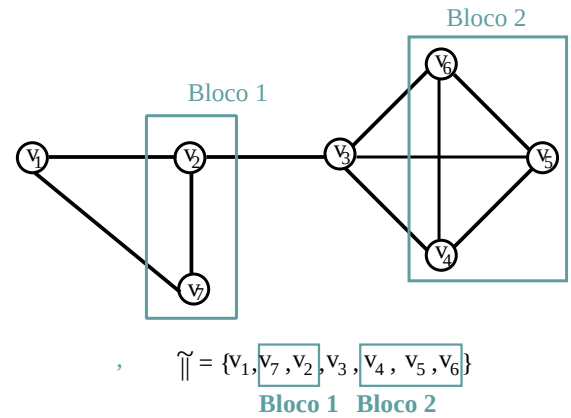

Figura A.5: Exemplo de blocos de um grafo G e de um esquema de eliminação bloco-contínuo

Os vizinhos dos vértices em B também formam uma cadeia. Na Figura A.6 veja um exemplo de um grafo cadeia. Para esse exemplo observe que $\tau(c) < \tau(b) \rightarrow V(c) \supseteq V(b)$, $\tau(c) < \tau(a) \rightarrow V(c) \supseteq V(a)$ e $\tau(b) < \tau(a) \rightarrow V(c) \supseteq V(a)$.

Dado um grafo bipartido $G(A \cup B, E)$ e uma ordenação τ de A, um grafo (G, τ)-*cadeia* é qualquer grafo bipartido $G'(A \cup B, E \cup E')$ para o qual τ é uma ordenação cadeia.

Para um dado grafo bipartido $G(A \cup B, E)$, o grafo $C(G)$ é um grafo formado pela adição de arestas para formar subgrafos completos em A e B.

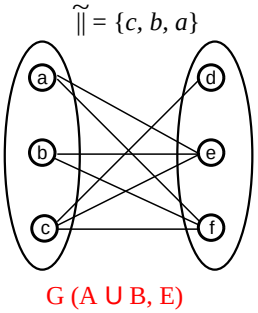

Figura A.6: Exemplo de uma ordenação em cadeia e de um grafo cadeia

Se V é o conjunto de vértices de um grafo G e τ é uma permutação de V, então definimos o *valor de corte linear com respeito a* π como

$$c_\tau(G) = \max_{1 \leq i \leq |V|} \{(u, v) \in E : \tau(u) \leq i < \tau(v)\}).$$

O *problema Arranjo de Corte Linear Mínimo* (Minimum Cut Linear Arrangement Problem - MCLA) é definido como a seguir: Dado um grafo $G = (V, E)$ e um inteiro positivo k, existe uma permutação τ de V tal que $c_\tau \leq k$?

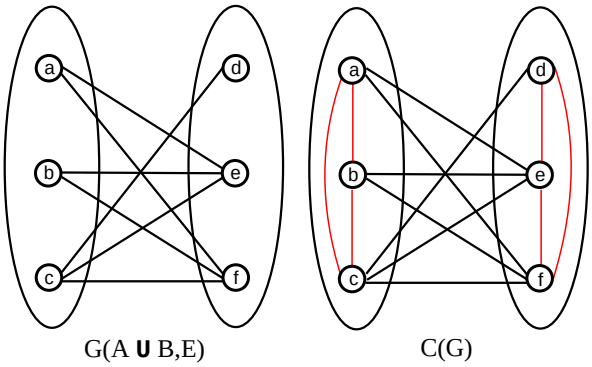

Figura A.7: Exemplo de um grafo $C(G)$

Garey and Johnson provaram que MCLA é um problema NP-Completo [13].

Na Figura A.8 é mostrado um exemplo de um grafo, e o valor do corte para a permutação τ dada. Considerando a permutação $\tau = \{a, b, c, d\}$ o maior valor de corte encontrado é igual a 3, e nesse caso, as arestas de corte são $\{e_1, e_2, e_3\}$. Veja:

Para $i = 1$	$(a, b) \rightarrow \tau(a) \leq 1 < \tau(b) \Rightarrow 1 \leq 1 < 2$ $(a, c) \rightarrow \tau(a) \leq 1 < \tau(c) \Rightarrow 1 \leq 1 < 3 \Rightarrow \lvert\{e_1, e_2, e_3\}\rvert = 3$ $(a, d) \rightarrow \tau(a) \leq 1 < \tau(d) \Rightarrow 1 \leq 1 < 4$
Para $i = 2$	$(b, c) \rightarrow \tau(b) \leq 2 < \tau(c) \Rightarrow 2 \leq 2 < 3$ $(b, d) \rightarrow \tau(b) \leq 2 < \tau(d) \Rightarrow 2 \leq 2 < 4 \Rightarrow \lvert\{e_4, e_6\}\rvert = 2$
Para $i = 3$	$(c, d) \rightarrow \tau(c) \leq 3 < \tau(d) \Rightarrow 3 \leq 3 < 4 \Rightarrow \lvert\{e_5\}\rvert = 1$

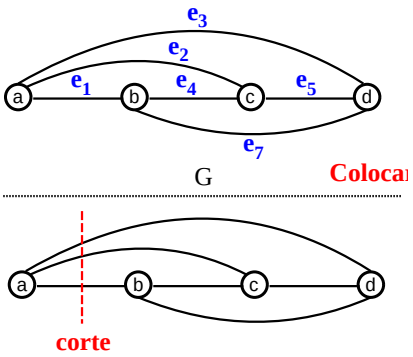

Figura A.8: Valor de corte para uma determinada permutação τ

Dado um grafo qualquer $G(V, E)$, Arnborg et al em [1] mostraram como construir um *grafo bipartido* $G'(A \cup B, E')$:

- Cada vértice $x \in V$ é representado por $\Delta(G) + 1$ vértices em A e $\Delta(G) - deg(x) + 1$ vértices em B. A_x denota os vértices em A que representam o vértice x. B_x denota os vértices em B que representam o vértice x.

- Cada aresta $e \in E$ é representada por dois vértices em B . B_e denota os vértices que representam a aresta e.
- As arestas em E' são dos seguintes tipos:
  - Todos os vértices em A_x são adjacentes a todos os vértices B_x.
  - Todos os vértices em A_x são adjacentes aos dois vértices em B_e se x é um extremo de e.

Como exemplo dessa construção, considere a Figura A.9.

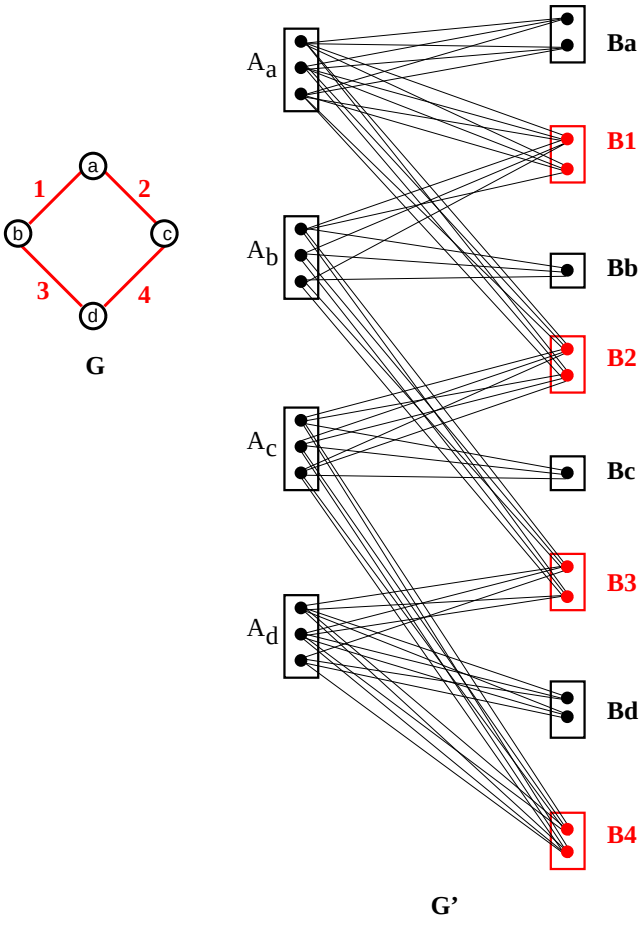

Figura A.9: Exemplo de construção do grafo bipartido $G'(A \cup B, E')$

A.3.2 Lemas, Teoremas e Corolários

Lema A.9 *[1] Todo grafo k-cordal é k-decomponível.*

Prova. Dado um grafo k-cordal G, pelo Lema 3.3, página 21, sabemos que G possui pelo menos um vértice simplicial u. Como $\omega(G) \leq k+1$, temos que $|V(u)| \leq k$. Sabemos

também que $V(u)$ é uma clique que separa u o resto do grafo. Além disso, $V'(u)$ é um grafo completo com no máximo $k+1$ vértices e $G-u$ trata-se também de um grafo cordal cuja maior clique terá tamanho no máximo $k+1$. Logo, temos que G é um grafo k-decomponível. ■

Veja o Lema A.10 para ver a relação entre $k_t(G)$ e $k_c(G)$.

Lema A.10 *[1] Para qualquer grafo G que não é um grafo completo, $k_t(G) = k_c(G)$.*

Prova.

1. $k_c(G) \leq k_t(G)$: Seja G' um supergrafo $k_t(G)$-árvore de G. Pelo Lema 3.5, 29, sabemos que uma k-árvore é também um grafo k-cordal. Sendo assim, G' é também $k_t(G)$-cordal. Logo, $k_t(G) \geqslant k_c(G)$.

2. $k_t(G) \leq k_c(G)$: Seja G' um supergrafo $k_c(G)$-cordal. Sabemos pelo Lema A.9 que G' é $k_c(G)$-decomponível. Logo, pelo Teorema 7 de [2] temos que G' também é uma $k_c(G)$-árvore. Logo, G é uma $k_c(G)$-árvore parcial, e $k_c(G) \geqslant k_t(G)$.

■

O Lema A.11, relaciona grafos cadeia e grafos cordal.

Lema A.11 *[32] Um grafo bipartido $G\big(A \cup B, E\big)$ é um grafo cadeia se, e somente se, $C(G)$ é cordal.*

Lema A.12 *[1] Seja H um supergrafo cordal minimal de G. Então existe uma ordenação de eliminação bloco-contínuo π tal que H é o grafo preenchido de G com respeito a π.*

Prova. Sabemos por [25] que um grafo é cordal se, e somente se, ele tem um esquema de eliminação perfeito, e que todo supergrafo cordal minimal em aresta de um grafo G é um grafo preenchido com respeito a algum esquema de eliminação. Assim, H é o grafo preenchido com respeito a algum esquema de eliminação. Um vértice é simplicial se, e somente se, todos os outros vértices de seu bloco são também simpliciais. Considerando que a eliminação de qualquer vértice preserva essa propriedade, qualquer grafo cordal tem um esquema perfeito de eliminação bloco-contínuo. Dado tal esquema, qualquer ordenação

dos vértices em bloco determina um esquema perfeito de eliminação bloco-contínuo. Se u e v pertencem ao mesmo bloco de G, então a adição da aresta de preenchimento implica na adição da aresta de preenchimento (u, v). Assim, os blocos de G formam um refinamento (possivelmente trivial) dos blocos de H. Agora, fazemos π o esquema perfeito de eliminação bloco-contínuo para H, que é também o esquema para G. ■

A seguir, os conceitos de grafo cadeia e caminho cordal são usados para provar que o problema k-árvore parcial é pelo menos tão difícil quanto o problema Arranjo de Corte Linear Mínimo. Para isso Arnborg and Proskurowski em [2] mostraram a relação entre o valor de corte linear de um grafo G e os valores de k' tal que $C(G')$ é uma k-árvore parcial.

Lema A.13 *[1] Dado um grafo G e um inteiro positivo k, G tem um valor de corte linear mínimo k com respeito a alguma permutação π se, e somente se, o grafo correspondente $C(G')$ é uma k'-árvore parcial para $k' = \big(\Delta(G) + 1(|V| + 1)\big) + k - 1$.*

Prova. Sabendo que uma k'-árvore é um grafo k'-cordal, segue do Lema A.12 que se $C(G')$ é uma k'-árvore parcial, então existe uma esquema de eliminação bloco-contínuo π'tal que nenhum vértice tem grau maior que k' quando eliminado. Seja Fo grafo preenchido de $C(G')$ com respeito a essa permutação π'. F é também supergrafo do grafo preenchido de $C(G')$ com respeito a qualquer ordem perfeita de eliminação de F. Mas sendo F cordal, suas arestas entre A e B formam um grafo cadeia pelo Lema A.11, e é fácil ver que F tem uma ordenação perfeita de eliminação iniciando com todos os vértices de A na ordem reversa à ordenação cadeia, que é também uma ordenação contínua nos blocos de A_v. Sem perda de generalidade, podemos assumir que π' é tal ordenação. Seja π a ordenação de blocos em A induzidos por π'. Assuma sem perda de generalidade que os vértices de G são numerados na ordem π e são identificados por seus números. Considere o grafo resultante da eliminação dos vértices nos primeiros $i - 1$ blocos de $C(G')$. Nesse grafo, cada vértice de A_i é adjacente a:

- Outros $\Delta(G)$ vértices em A_i;
- $\Delta(G) + 1$ vértices em cada um dos $A_{i+1}, ..., A_{|V|}$
- $\Delta(G) + 1 - deg_G(j)$ vértices em B_j para $j = 1, ..., i$ (essas são arestas de preenchimento, exceto para $j = i$);

- E a dois vértices em B_e para cada aresta e incidente a pelo menos um vértice em $\{1, ..., i\}$ essas são arestas de preenchimento para e não incidente a i).

Essas adjacências somam

$$\Delta(G) + \big(\Delta(G)+1\big)\big(|V|-i\big) + \left[\big(\Delta(G)+1\big) \times i - \Big(\sum_{j=1}^{i} deg_G(j)\Big)\right] + 2|E_1^i| + 2|E_2^i|$$

Onde E_1^i é o conjunto de arestas com exatamente um vértice em $\{1, ..., i\}$, e E_2^i é o conjunto de arestas com ambos os vértices em $\{1, ..., i\}$. Obviamente, $\sum_{j=1}^{i} deg_G(j) = 2|E_1^2| + |E_1^i|$ então o grau de um vértice em A_i pode ser simplificado para:

$$\Delta(G) + \big(\Delta(G)+1\big)\big(|V|-i\big) + \left[\big(\Delta(G)+1\big) \times i - \Big(\sum_{j=1}^{i} deg_G(j)\Big)\right] + 2|E_1^i| + 2|E_2^i|$$

$$\Delta(G) + \big(\Delta(G)+1\big)\big(|V|-i\big) + \left[\big(\Delta(G)+1\big) \times i - \Big(2|E_2^i| + |E_1^i|\Big)\right] + 2|E_1^i| + 2|E_2^i|$$

$$\Delta(G) + \big(\Delta(G)+1\big)\big(|V|-i\big) + \left[\big(\Delta(G)+1\big) \times i - 2|E_2^i| - |E_1^i|\right] + 2|E_1^i| + 2|E_2^i|$$

$$\Delta(G) + \big(\Delta(G)+1\big)\big(|V|-i\big) + \big(\Delta(G)+1\big) \times i - 2|E_2^i| - |E_1^i| + 2|E_1^i| + 2|E_2^i|$$

$$\Delta(G) + \big(\Delta(G)+1\big)\big(|V|-i\big) + \big(\Delta(G)+1\big) \times i - |E_1^i| + 2|E_1^i|$$

$$\Delta(G) + \big(\Delta(G)+1\big)\big(|V|-i\big) + \big(\Delta(G)+1\big) \times i + |E_1^i|$$

$$\big(\Delta(G)+1\big) + \big(\Delta(G)+1\big)\big(|V|-i\big) + \big(\Delta(G)+1\big) \times i + |E_1^i| - 1$$

$$\big(\Delta(G)+1\big)\big(1 + (|V|-i) + i\big) + |E_1^i| - 1$$

$$\big(\Delta(G)+1\big)\big(1 + |V|\big) + |E_1^i| - 1$$

Sabendo que E_1^i é o número de arestas entre os vértices em $\{1, ..., i\}$ e os vértices em $\{i+1, ..., |V|\}$, particularmente na ordenação π, o tamanho máximo de E_1^i sobre todos os $i's$ é o valor do corte linear de G com respeito a π. Esse valor também determina o valor máximo de uma clique em $C(G')$. Assim, mostramos que a k'-cordalidade implica na existência de um arranjo linear com valor de corte k. Inversamente, a existência de uma ordenação π com respeito a, do qual G tem um valor de corte linear k implica que a maior clique em F, do grafo preenchido de $C(G')$ com respeito a π', tem tamanho $k'+1$ (pela verificação de uma ordenação induzida π' de seus vértices). Isso completa a prova. ■

Na Figura A.10 está um exemplo de um grafo $C(G')$ preenchido. As arestas de

preenchimento estão em negrito. Observe que cada A_i é adjacente a todos os vértices a que seus vizinhos A_j, $j = \{1, ..., i-1\}$, são adjacentes. Assim, podemos verificar que temos uma ordenação perfeita de eliminação começando pelos vértices de A e eliminando sequencialmente cada A_i, $i = \{1, ..., A_{|V|}\}$.

Na Figura A.11 veja as adjacências de um vértice A_i no momento de sua eliminação.

Teorema A.6 *O problema k-árvore parcial é NP-Completo.*

Prova. (Dificuldade para NP) Segue do Lema A.13 e do fato de que $C(G')$ pode ser construído de G em tempo polinomial. (Membro de NP): Por uma apropriada (não-determinística) escolha da ordem dos vértices, o processo de eliminação é facilmente transformado em uma verificação de tempo polinomial que um grafo é uma k-árvore parcial. ■

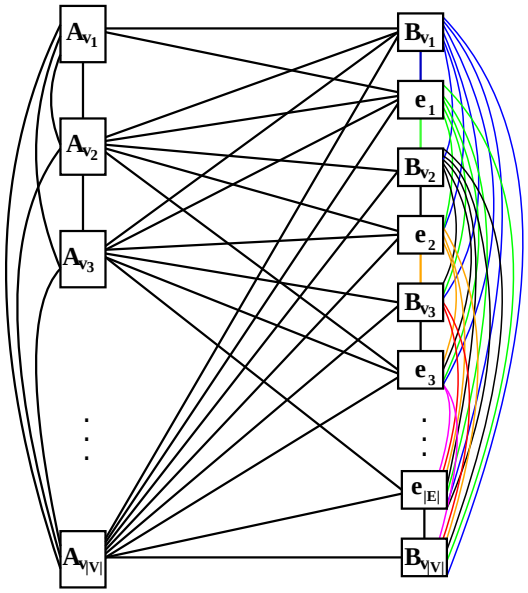

Figura A.10: Um grafo $C(G')$ preenchido

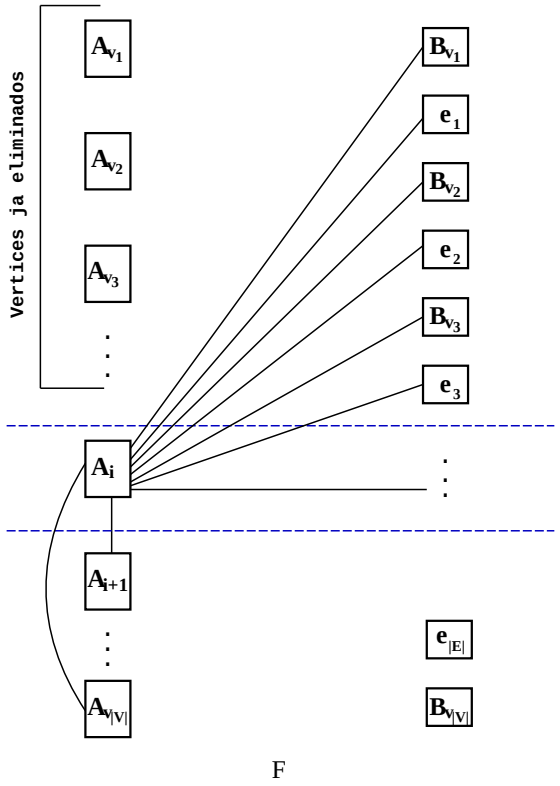

Figura A.11: Adjacências de um vértice A_i no momento de sua eliminação

Referências

[1] S. Arnborg, D. G. Corneil, and A. Prokurowski. Complexity of finding embedding in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, 8:277–284, 1987.

[2] S. Arnborg and A. Prokurowski. Characterization and recognition of partial 3-trees. *SIAM Journal on Algebraic and Discrete Methods*, 7:305–314, 1986.

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

[4] T. Biedl. Graph-theoretic algorithms - manuscript, 2005.

[5] H. L. Bodlaender. Some classes of graphs with bounded treewidth. *Bulletin of the EATCS*, 36:116–125, 1988.

[6] H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996.

[7] H. L. Bodlaender and T. Kloks. Better algorithms for the pathwidth and treewidth of graphs. In *Proceedings of the 18th international colloquium on Automata, languages and programming*, pages 544–555, New York, NY, USA, 1991. Springer-Verlag New York, Inc.

[8] H. L. Bodlaender and T. Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *J. Algorithms*, 21(2):358–402, 1996.

[9] L. E. X. Carvalo. Decomposição em árvore de grafos com largura limitada - uma pesquisa algorítmica. Dissertação de mestrado, Universidade Federal do Ceará, 2002.

[10] G. A. Dirac. On rigid circuit graphs. Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg 25, Universität Hamburg, 1961.

[11] J. Edmonds. Paths, trees, and flowers. *Canad. J. Math.*, 17:449–467, 1965.

[12] D. R. Fulkerson and O. A. Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15(15):835–855, 1965.

[13] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[14] F. Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory*, 16:47–56, 1974.

[15] L. A. Goldberg, S. Kelk, and M. Paterson. The complexity of choosing an h-colouring (nearly) uniformly at random. In *STOC '02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 53–62, New York, NY, USA, 2002. ACM.

[16] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York, 1980.

[17] P. Hell, S. Klein, L. T. Nogueira, and F. Protti. Partitioning chordal graphs into independent sets and cliques. *Discrete Appl. Math.*, 141(1-3):185–194, 2004.

[18] T. Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994.

[19] A. M. C. A. Koster, H. L. Bodlaender, and Stan P. M. Van Hoesel. Treewidth: Computational experiments. In *Electronic Notes in Discrete Mathematics*, pages 54–57. Elsevier Science Publishers, 2001.

[20] C. G. Lekkerkerker and J. C. Boland. Representation of a finite graph by a set of intervals on the real line. *Fundamenta Mathematicae*, 51:45–64, 1962.

[21] J. Matoušek and R. Thomas. Algorithms finding tree-decompositions of graphs. *J. Algorithms*, 12(1):1–22, 1991.

[22] R. H. Mshring and H. L. Bodlaender. The pathwidth and treewidth of cographs. *SIAM Journal on Discrete Mathematics*, 6(6):181–188, 1993.

[23] N. Robertson and P. D. Seymour. Graph minors. I. Excluding a forest. *Journal of Combinatorial Theory*, 35:39–61, 1983.

[24] N. Robertson and P. D. Seymour. Graph minors II: algorithmic aspects of tree-width. *Journal Algorithms*, 7:309–322, 1986.

[25] D. J. Rose. Triangulated graphs and the elimination process. *Journal of Mathematical Analysis and Applications*, 32:597–609, 1970.

[26] D. J. Rose. On simple characterization of k-trees. *Discrete Mathematics*, 7:317–322, 1974.

[27] D. J. Rose, R. E. Tarjan, and G. S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing*, 5:266–283, 1976.

[28] A. S. F. Silva. Um estudo computacional sobre o problema de decomposição de grafos em árvore. Dissertação de mestrado, Universidade Federal do Ceará, 2005.

[29] J. L. Szwarcfiter. *Grafos e algoritmos computacionais*. Editora Campus, 1986.

[30] Jan van Leeuwen. Graph algorithms. In *Handbook of Theoretical Computer Science, A: Algorithms and Complexity Theory*, pages 527–631, Amsterdam, 1990. North Holland Publ. Comp.

[31] J. A. Wald and C. J. Colbourn. Steiner trees, partial 2-trees, and minimum IFI networks. *Networks*, 13:159–167, 1983.

[32] M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM Journal on Discrete Mathematics*, 2:77–79, 1981.