UNIVERSIDADE FEDERAL FLUMINENSE

Átila Arueira Jones

Gerador de Cografos com Atraso Linear e Partição de Arestas em Cliques em Grafos (k, ℓ)

NITERÓI

2018

UNIVERSIDADE FEDERAL FLUMINENSE

Átila Arueira Jones

Gerador de Cografos com Atraso Linear e Partição de Arestas em Cliques em Grafos (k, ℓ)

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Doutor em Computação. Área de concentração: Algoritmos e Otimização

Orientador:
Fábio Protti

Co-orientador:
Renata Raposo Del-Vecchio

NITERÓI

2018

Ficha catalográfica automática - SDC/BEE

J76g Jones, Átila Arueira
Gerador de Cografos com Atraso Linear e Partição de Arestas em Cliques em Grafos (k,l) / Átila Arueira Jones ; Fábio Protti, orientador ; Renata Raposo Del-Vecchio, coorientadora. Niterói, 2018.
125 f. : il.

Tese (doutorado)-Universidade Federal Fluminense, Niterói, 2018.

DOI: http://dx.doi.org/10.22409/PGC.2018.d.12736803752

1. Grafo. 2. Algoritmo em grafos. 3. Produção intelectual. I. Título II. Protti,Fábio, orientador. III. Raposo Del-Vecchio, Renata, coorientadora. IV. Universidade Federal Fluminense. Escola de Engenharia.

CDD -

Bibliotecária responsável: Fabiana Menezes Santos da Silva - CRB7/5274

Átila Arueira Jones

Gerador de Cografos com Atraso Linear e
Partição de Arestas em Cliques em Grafos (k,l)

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Doutor em Computação. Área de concentração: Algoritmos e Otimização

Aprovada em novembro de 2018.

BANCA EXAMINADORA

Prof. Fábio Protti - Orientador, UFF

Profª. Renata Raposo Del-Vecchio - Co-orientador, UFF

Prof. Luerbio Faria, UERJ

Profª. Maria Aguieiras Alvarez de Freitas, UFRJ

Prof. Uéverton dos Santos Souza, UFF

Prof. Yuri Abitbol de Menezes Frota, UFF

Niterói
2018

À minha incrível esposa Allana.

Agradecimentos

Antes de tudo quero agradecer imensamente aos meus orientadores Fábio Protti e Renata Del-Vecchio por guiarem os estudos sempre da melhor forma possível, sempre com muito empenho e sabedoria. São exemplos de pesquisadores e professores que carregarei para vida toda. "*Se eu vi mais longe, foi por estar sobre ombros de gigantes.*", Isaac Newton.

À minha esposa Allana, agradeço primeiramente pelo carinho e paciência fornecida a cada dia, aquela que me acompanha bem de perto toda a minha carreira acadêmica desde o primeiro período da graduação. Obrigado por sempre acreditar e me incentivar a seguir em frente. A sua contribuição na elaboração desse trabalho é imensurável.

Agradeço aos meus pais, por estarem ao meu lado por todos esses anos de muito amor e incentivo nos estudos. Se sou alguém hoje... é tudo graças a vocês.

Muito obrigado a todos os professores do Núcleo de Matemática do IF sudeste MG, que me apoiaram e se dispuseram a ajudar a reduzir minha carga de trabalho nos momentos finais do doutorado.

Felizmente também pude contar com amigos e outros familiares, que todos de forma direta ou indireta contribuíram na elaboração deste trabalho. Mas direciono um agradecimento especial ao Marcos, Robert e Allícia.

Finalmente agradeço a CAPES pelo apoio financeiro a este trabalho.

Resumo

Esta tese aborda dois diferentes problemas. O primeiro é focado na classe dos cografos, onde usamos a representação de coárvores para desenvolver um algoritmo capaz de gerar, sem isomorfismos, todos os cografos com n vértices, cujo tempo de geração entre dois cografos consecutivos foi feito em tempo $O(n)$. Em seguida aplicamos este procedimento para conjecturar uma nova cota para o número b-cromático de um cografo conexo, dada em termos do raio espectral. O segundo problema abordado é o estudo da complexidade do problema de obter o menor número de cliques disjuntas em arestas necessárias para particionar as arestas de G, chamado $cp(G)$. Este problema já é conhecido como $\mathcal{NP}$-completo para grafos em geral, inclusive para *split*, grafos K_4-livres e grafos cordais. Provamos que a NP-completude é mantida para a classe dos grafos 3-coloríveis, subclasse dos K_4-livre. Além disso, conseguimos obter a complexidade do problema para grafos (k, ℓ), aqueles cujo conjunto de vértices pode ser particionado em k independentes e ℓ cliques, estabelecendo quando o problema é $\mathcal{NP}$-completo ou polinomial para cada k e ℓ fixados.

Palavras-chave: cografo, gerador, partição de arestas em cliques, grafo-(k, ℓ).

Abstract

This thesis approaches on two different problems. The first one is focused on the class of cographs, where we use the representation of cotrees to develop an algorithm capable of generating, without isomorphisms, all the n vertex cographs and whose generation time between two consecutive cografos was done in time $O(n)$. Then we apply this procedure to conjecture a new bound for the b-chromatic number of a connected cograph given in terms of the spectral radius. The second problem is the study of the complexity of the problem of obtaining the least number of disjoint cliques on edges necessary to partition the edges of G, called $cp(G)$. This problem is already known as $\mathcal{NP}$-complete for general graphs, including *split*, K_4-free graphs and chordal graphs. We prove that NP-completeness is maintained for the class of 3-colorable graphs, subclass of K_4-free. In addition, we get the complexity of the problem for graphs (k, ℓ), those whose set of vertices can be partitioned into k independent and ℓ cliques, establishing when the problem is $\mathcal{NP}$-complete or polynomial for each fixed k and ℓ.

Keywords: cograph, generator, edge clique partition, (k, ℓ)-graph.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

ECP	:	Partição de arestas em cliques;
ECC	:	Cobertura de arestas em cliques;
MTED	:	Maior número de triângulos disjuntos em arestas;
E3C	:	Cobertura 3-exata;
MIS	:	Conjunto independente máximo;
VCP	:	Partição de vértices em cliques;

Sumário

Capítulo 1

Introdução

Os grafos são estruturas fundamentais em diversas áreas, em especial Teoria da Computação e Matemática. No campo teórico sua importância é mostrada pelo alto número de publicações na área, com estudos acerca de estruturas, parâmetros e solução de problemas computacionais. Por outro lado, temos incontáveis situações reais que podem ser modeladas por grafos, como redes, logística de transporte, redes sociais, fluxogramas, dentre vários.

Esta tese é focada em dois diferentes problemas: criação de um algoritmo capaz de gerar cografos de forma eficiente e o problema de partição das arestas de um grafo em cliques para grafos (k, ℓ). Em ambos os casos apresentamos novas contribuições.

Quanto ao primeiro problema, os *cografos* foram introduzidos por volta de 1970 por diferentes autores, como Lerchs [39]. Em geral os cografos são mais conhecidos como os grafos livres de P_4, definição alternativa introduzida por Corneil *et al* [12]. Problemas provados como $\mathcal{NP}$-completo para grafos em geral, apresentam complexidade polinomial quando restritos a esta classe, dentre estes podemos citar: clique, isomorfismo e coloração (e variações) [12] [6]. Os cografos possuem propriedades interessantes tais como são grafos perfeitos [12], L-integrais [44], b-contínuos [6], dentre outros. Desde então foram introduzidas diversas classes que podem ser vistas como generalização dos cografos, como os grafos de *Distância Hereditária*, *grafos (q,q-4)*, P_4-esparso, P_4-extensíveis e P_4-redutíveis.

A área de combinatória enumerativa tem a finalidade de obter todas as maneiras que podemos formar um certo padrão, em Teoria dos Grafos o interesse é obter um algoritmo capaz de gerar todos os grafos com certa propriedade. Até então não é encontrado na literatura nenhum trabalho sobre geração de cografos, que é exatamente o tema desenvolvido neste trabalho. Mais especificamente, dado um inteiro n desenvolvemos um algoritmo

eficiente capaz de gerar, sem repetição, todos cografos com n vértices, cuja eficiência é obtida ao provarmos que o tempo de geração entre dois cografos consecutivos é linear em n. O único trabalho encontrado que estuda os cografos do ponto de vista da combinatória enumerativa é [50], contudo neste é mostrado apenas uma estimativa assintótica para o número de cografos possíveis, enquanto que no nosso trabalho somos capazes de gerar todos os cografos e além disso obter o número exato de cografos com n vértices.

São naturais as aplicações do nosso algoritmo, pois a geração de todos os cografos auxilia na formulação de novos resultados, em especial o procedimento também se mostra útil para encontrar contra-exemplos ou até validar conjecturas acerca dos cografos. O ponto que impulsionou o desenvolvimento deste algoritmo de geração dos cografos foi a desigualdade estimada na dissertação [30] que relaciona a b-coloração e Teoria Espectral de Grafos, duas áreas até então ainda não relacionadas na literatura. Então a primeira aplicação do nosso algoritmo ocorre neste próprio trabalho, onde implementamos e executamos testes computacionais em busca de contra-exemplos para a desigualdade estimada.

O segundo problema tratado nesta tese é a partição das arestas de um grafo em cliques, cujo primeiro resultado foi em 1966 por Erdós, Goodman e Pósa [17]. Dado G definimos o número $cp(G)$ como o tamanho da menor partição das arestas em cliques, cujo problema de decisão (ECP) é instanciado por um grafo G e um inteiro k onde é questionado se $cp(G) \leq k$. Em paralelo é definida a menor cobertura das arestas em cliques por $cc(G)$ e seu respectivo problema de decisão (ECC), vale ressaltar que a diferença entre as definições é que cobertura permite que as cliques possuam arestas em comum. Estes problemas possuem aplicações práticas, como o ECP que apresenta importância em computação biológica na classificação de clones de DNA, como visto em [18]; enquanto que ECC é aplicado em problemas de conflito de palavras-chave e faseamento de tráfego como citado em [51].

Um dos focos deste trabalho é o estudo da complexidade de ECP em determinadas classes de grafos. Em 1977 foi provado em [46] que os problemas ECP e ECC são $\mathcal{NP}$-completo para grafos gerais, já em 1988 [41] foi provado que ECP permanece $\mathcal{NP}$-completo para a classes dos grafos livres de K_4 e a classe dos cordais. No decorrer do texto veremos a complexidade do problema em outras classes. O nosso interesse é na complexidade do ECP para grafos (k, ℓ), aqueles cujo conjunto de vértices pode ser divido em k independentes e ℓ cliques. Wallis e Ju-Lin [56] em 1991, mostram que ECP é $\mathcal{NP}$-completo para grafos split, o que motivou este estudo. É natural que para k, ℓ arbitrários o problema seja $\mathcal{NP}$-completo , então um dos objetivos deste trabalho é obter a complexidade do problemas

para todos os possíveis valores de (k, ℓ) mostrando para quais valores de k, ℓ o problema deixa de ser polinomial e passar a ser $\mathcal{NP}$-completo . Uma das principais contribuições deste trabalho é o resultado que garante que ECP é $\mathcal{NP}$-completo para grafos 3-colorível (caso $k = 3$ e $\ell = 0$), feito através de uma redução polinomial. Sua importância se deve ao fato de apresentar uma subclasse de K_4-livre onde o problema permanece $\mathcal{NP}$-completo , resultado de 1988 publicado por Ma, Wallis e Ju-Lin [41], citado anteriormente. Além de conseguirmos determinar a complexidade de ECP para todos possíveis k, ℓ exibimos uma subclasse do $(3, 0)$ onde o problema se torna polinomial.

Esta tese está estruturada da seguinte forma:

No Capítulo seguinte apresentamos conceitos básicos, necessários para compreensão do texto.

No Capítulo 3 desenvolvemos o algoritmo gerador de cografos e toda teoria necessária para a elaboração do procedimento. Tanto o algoritmo quanto os principais resultados desenvolvidos neste capítulo foram publicados em *Theoretical Computer Science*, como visto em [32].

No Capítulo 4 trataremos da aplicação que motivou este trabalho. Inicialmente será feito um resumo da teoria necessária para o entendimento da desigualdade estimada em [30], e em seguida descreveremos os algoritmos implementados, incluindo os desenvolvidos no capítulo anterior e, por fim, apresentaremos os resultados computacionais após a execução dos testes. Tais resultados foram apresentados no *XLIX Simpósio Brasileiro de Pesquisa Operacional* e publicado nos anais do congresso, veja em [31].

No Capítulo 5 abordamos o problema de particionar as arestas de um grafo em cliques. Onde foi feita uma revisão da literatura sobre o problema e em seguida estudamos sua complexidade na classe dos grafos (k, ℓ). Os resultados obtidos neste capítulo estão em fase de submissão de artigo para revista.

No último capítulo concluímos o trabalho evidenciando os principais resultados obtidos.

Capítulo 2

Preliminares

2.1 Teoria dos Grafos

Seja $G(V, E)$ um grafo. Denotaremos a adjacência entre dois vértices u e v por $(u, v) \in E$, ou simplesmente $uv \in E$. O **grau** de um vértice v é o número de vértices adjacentes a v, cujo valor é denotado por $d(v)$, o grau mínimo, médio e máximo de um grafo são denotados por $\delta(G)$, $\overline{d}(G)$ e $\Delta(G)$, respectivamente, onde $\overline{d}(G) = \frac{\sum d(v)}{|V(G)|}$.

Denotamos por K_n o grafo **completo** de n vértices, ou seja, aquele onde qualquer par de vértices são adjacentes, em particular K_3 é comumente chamado de **triângulo**. Uma sequência finita de vértices distintos $v_{i_1}v_{i_2}\ldots v_{i_k}$ de G é chamado **caminho** de v_{i_1} à v_{i_k} se $(v_{i_l}, v_{i_{l+1}}) \in E$ para $1 \leq i \leq k-1$. Nas mesmas condições a sequência $v_{i_1}v_{i_2}\ldots v_{i_k}v_{i_1}$ é chamada de **ciclo**. O comprimento de um **caminho** ou **ciclo** é o número de arestas que neles ocorrem.

O grafo cujo conjunto de vértices é $V = \{v_1, \ldots, v_n\}$ e arestas $E = \{(v_i, v_{i+1}); 1 \leq i \leq n-1\}$ é chamado *grafo caminho* e denotado por P_n. De forma semelhante, o grafo ciclo C_n tem conjunto de arestas $E = \{(v_i, v_{i+1}); 1 \leq i \leq n-1\} \cup \{(v_n, v_1)\}$.

Exemplo 2.1.1. *Na ilustração abaixo temos os grafos C_6 e P_5.*

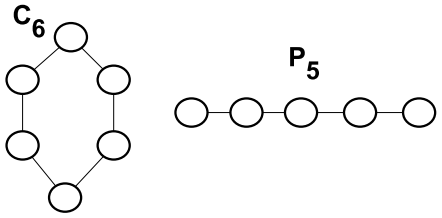

Figura 2.1: Exemplo de grafos ciclo e caminho

Se existe um caminho entre qualquer par de vértices do grafo, dizemos que o grafo é **conexo**, caso contrário é dito **desconexo**.

Diremos que $H(V', E')$ é um **subgrafo** de G quando $V' \subseteq V$ e $E' \subseteq E$. No caso em que $(x, y) \in E'$ se, e somente se, $(x, y) \in E$, diremos que H é **subgrafo induzido** de G, ou ainda G induz H, neste caso escrevemos $H = G[V']$ ou $H = G[E']$. Um grafo G é livre de H se o grafo H não é subgrafo induzido de G.

Exemplo 2.1.2. *Na figura abaixo vemos G e seu subgrafo induzido $H = G[\{v_1, v_2, v_4, v_6\}]$, observe ainda que G é livre de C_6.*

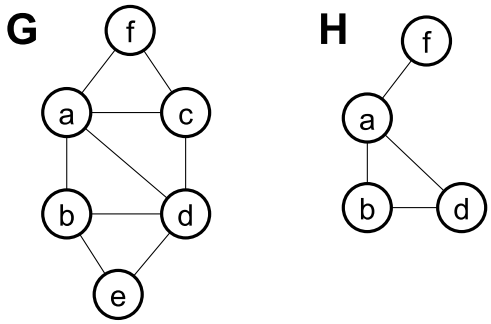

Figura 2.2: Exemplo de subgrafos de G

Conforme a literatura, dado um grafo $G(V, E)$, diremos que um subconjunto de vértices $S \subset V$ é estável se o subgrafo induzido $G[S]$ não possui arestas, enquanto que $C \subset V$ é dito uma p-clique se $G[S]$ é isomorfo ao grafo completo K_p. O número clique $\omega(G)$ de um grafo é o tamanho da maior clique induzida em G, analogamente o número independente $\alpha(G)$ é o tamanho do maior conjunto estável. No grafo G ilustrado no Exemplo acima temos $\omega(G) = 3$ e $\alpha(G) = 2$.

Também definimos algumas operações em grafos:

- O grafo *complementar* de G é o grafo $\overline{G}(V', E')$, onde $V' = V$ e $E' = \{(x, y); (x, y) \notin E\}$. É fácil ver que $\overline{\overline{G}} = G$;
- A *união* de $G_1(V_1, E_1)$ e $G_2(V_2, E_2)$, denotada por $G_1 \cup G_2$, é o grafo cujo conjunto de vértices é $V(G_1) \cup V(G_2)$ e arestas $E(G_1) \cup E(G_2)$. A união de k cópias de um grafo G é denotada por kG;
- O *join* de $G_1(V_1, E_1)$ e $G_2(V_2, E_2)$, também conhecida como *união complementar*, é o grafo $G_1 \vee G_2$ com conjunto de vértices $V = V_1 \cup V_2$ e arestas $E = E_1 \cup E_2 \cup \{(x, y); x \in E_1 \text{ e } y \in E_2\}$. Observe que vale $\overline{G_1 \cup G_2} = \overline{G_1} \vee \overline{G_2}$

Exemplo 2.1.3. *Na figura abaixo vemos o join de dois grafos.*

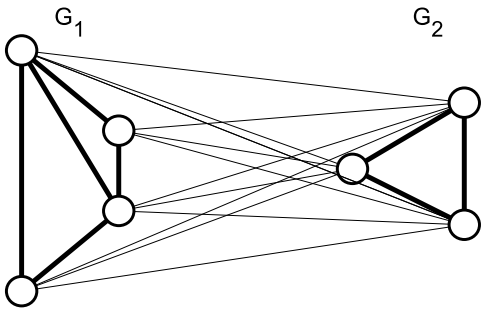

Figura 2.3: $G_1 \vee G_2$

Dois grafos G_1 e G_2 são ditos isomorfos, denotado por $G_1 \cong G_2$, quando podemos obter um do outro somente por uma permutação de vértices, ou melhor, existe uma aplicação $f : V(G_1) \to V(G_2)$ biunívoca de modo que $(u, v) \in E(G_1)$ se, e somente se, $(f(u), f(v)) \in E(G_2)$.

2.2 Árvore

Os grafos conexos que não possuem nenhum ciclo induzido são chamados de **árvore**. Observe que uma árvore é o grafo com o menor número de arestas tal que este seja conexo, uma vez que cada par de vértices são ligados por um único caminho, cuja prova pode ser vista em [5]. Uma árvore é dita enraizada se um vértice qualquer é elegido como raiz. A seguir introduzimos algumas nomenclaturas básicas que usaremos no decorrer do texto. Dada uma árvore T:

- O vértice raiz é denotado por $raiz(T)$.
- Se T não é trivial, os vértices cujo grau é 1, são chamados *folhas*, os demais são chamados *vértices internos*;
- Dado um vértice v (diferente da raiz) denotamos P_v o único caminho de v até a raiz;
- Os vértices de P_v (exceto v) são ditos **ancestrais** de v em T, onde o antecessor imediato de v em P_v é chamado **pai** de v e denotado por $pai_T(v)$. Fixamos $pai(raiz(T)) =$ ***null*** .
- Para cada vértice interno v definimos seus **filhos** pelo conjunto $filhos_T(v) = \{u \in V(T) | pai_T(u) = v\}$

- Se v não é a raiz, definimos sua **irmandade** por $I_T(v) = \{filhos_T(pai_T(v))\}$. Fixamos $I_T(raiz(T)) = \{raiz(T)\}$;
- Para cada vértice v definimos seu nível de forma recursiva:

$$nivel_T(raiz(T)) = 0 \qquad nivel_T(v) = nivel_T(pai(v)) + 1 \ ;$$

- O nível da árvore é dado por $nivel(T) = max\{nivel_T(v); v \in V(T)\}$ e os vértices de nível i dados por $nivel(T, i) = \{v \in V(T);\ nivel_T(v) = i\}$.
- Dado $v \in V(T)$ definimos por $T(v)$ a subárvore induzida por T cuja raiz é v.
- Dados v, w vértices distintos em T, o último ancestral comum (*last common ancester*) de v e w é o vértice de maior nível comum em ambos caminhos P_v e P_w, denotado por $lca(v, w)$.

Para não sobrecarregar o texto, em casos livres de ambiguidade ocultaremos o índice T das notações acima.

Exemplo 2.2.1. *Abaixo temos o exemplo de uma árvore enraizada em a. A partir desta árvore podemos concluir, por exemplo:* $pai(g) = c$; $pai(a) =$ ***null*** ; $lca(m, f) = c$; $I(g) = \{f, g, h\}$; $filhos(a) = \{b, c, d\}$, *os ancestrais de o são* h, c, a; $nivel(f) = 2$; $nivel(T, 3) = \{k, l, m, n, o\}$; $nivel(T) = 4$.

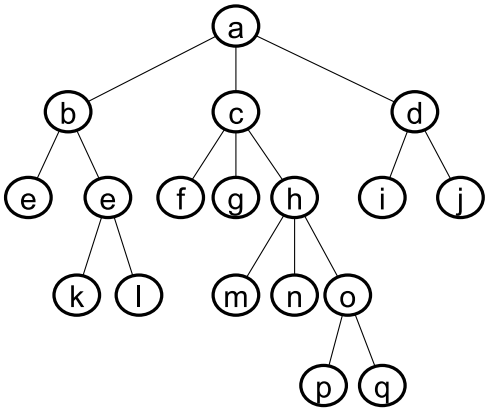

Figura 2.4: Exemplo de árvore

2.3 Cografo

Um dos focos deste trabalho é a classe dos cografos. Nesta seção introduziremos sua definição e trataremos alguns conceitos básicos conhecidos na literatura e que serão úteis

no decorrer do trabalho.

Os cografos são conhecidos por aqueles livres de P_4, então tal classe contém os *thresholds*, que são os grafos livres de $2K_2$, P_4 e C_4. Apesar dos cografos serem lembrados pela proibição de P_4, sua definição original é feita de forma recursiva [12]:

- O grafo trivial é cografo;
- Se G é cografo então $\overline{G}$ é cografo
- Se G_1 e G_2 são cografos então $G_1 \cup G_2$ é cografo.

Como vale a relação $\overline{G_1 \cup G_2} = \overline{G_1} \vee \overline{G_2}$, então um cografo pode ser obtido através de sucessivas operações de *join* e união entre cografos. Corneil *et al* [12] mostraram que um cografo pode ser representado por uma árvore, chamada coárvore, da seguinte forma: as folhas da coárvore são os vértices do cografo e os vértices internos de T representam operações de *join* ou união entre as árvores induzidas pelos seus filhos, recebendo nomenclaturas *tipo*-1 e *tipo*-0, respectivamente. O que assegura que cada nó interno possua ao menos dois filhos. Além disso, a coárvore é definida de forma que os filhos de vértices *tipo*-1 serem folhas ou nós internos do *tipo*-0, e vice-versa. Note que cada coárvore é referente a um único cografo, assim como cada cografo tem somente uma coárvore associada, a menos de permutação entre vértices. Dado um grafo G sua coárvore é denotada por $T(G)$.

Exemplo 2.3.1. *Na figura abaixo temos a coárvore $T(G)$ referente ao cografo G.*

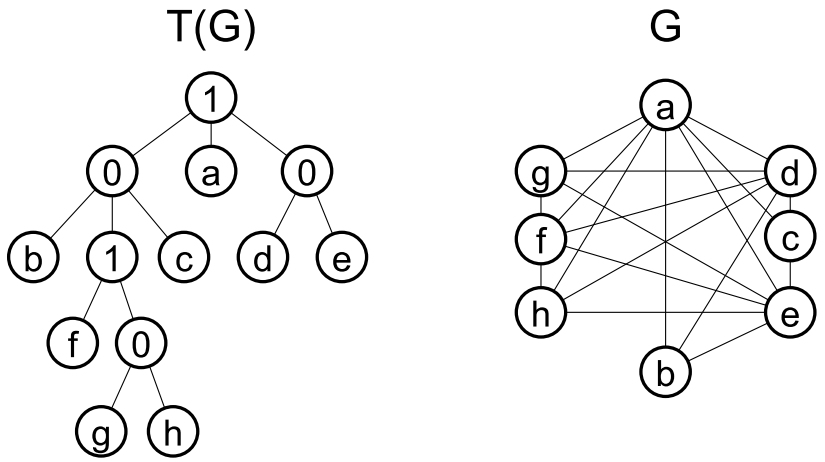

Figura 2.5: Coárvore $T(G)$ e seu respectivo cografo G

Os seguintes fatos são imediatos da definição de coárvore, e podem ser encontrados em [12].

Fato 2.3.1. *[12] Considere G um cografo.*

- *Um par de vértices v, w de G são adjacentes se, e somente se, o ancestral comum das suas folhas correspondentes em $T(G)$ é um vértice tipo-1.*
- *Um cografo é conexo se, e somente se, a raiz de sua coárvore é tipo-1.*
- *Dado um vértice v da coárvore $T(G)$, então $T(v)$ é a coárvore associada a um cografo induzido de G pelos vértices de $T(v)$.*

2.4 Grafos (k, ℓ)

Dados dois inteiros $k, \ell \geq 0$ diremos que um grafo é (k, ℓ) se seu conjunto de vértices pode ser particionado em k conjuntos independentes e ℓ cliques, esta é outra classe que será focada neste trabalho, mais especificamente no capítulo 5.

Exemplo 2.4.1. *O grafo abaixo é* $(2, 1)$.

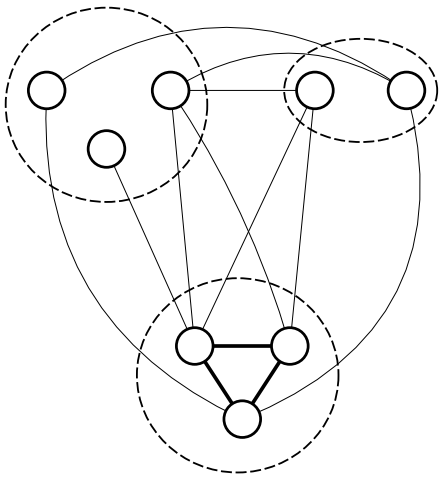

Figura 2.6: Exemplo grafo (2,1)

Como podemos considerar conjuntos vazios nas partições do conjunto de vértices, garantimos o seguinte fato:

Fato 2.4.1. *Se $\ell_2 \geq \ell_1 \geq 0$ e $k_2 \geq k_1 \geq 0$ são inteiros, então a classe dos grafos (k_1, ℓ_1) está contida na classe dos grafos (k_2, ℓ_2).*

Vale também observar que a noção de grafos (k, ℓ) pode ser vista como uma generalização de classes já conhecidas na literatura. Em particular quando $\ell = 0$, a classe representa

os grafos k-coloríveis cores; quando $k = \ell = 1$ o grafo é chamado de *split*; quando $k = 0$ e $\ell = 2$ pode ser chamado de *co-bipartido*, uma vez que é obtido do complemento de um grafo bipartido.

É conhecido que para $k \geq 3$ ou $\ell \geq 3$ é $\mathcal{NP}$-completo reconhecer se um grafo é (k, ℓ), conforme provado em [7]. Podemos encontrar na literatura diversos trabalhos acerca desta classe, como em [14] onde é estudado o Problema Sanduíche para Grafos (k, ℓ), ou alguns focados em estudar o comportamento em algumas subclasses como cordal-(k, ℓ) [23], cografo-(k, ℓ) [16], P_4-esparso-(k, ℓ) [8], dentre outras.

Capítulo 3

Geração de cografos

Neste capítulo desenvolveremos um algoritmo eficiente capaz de gerar todos os cografos com n vértices. Todos resultados e definições desenvolvidos neste capítulo são contribuições desta tese, salvo aqueles com referência explícita. Inicialmente, introduziremos novos conceitos e obteremos resultados intermediários para alcançarmos o desejado.

Este capítulo foi desenvolvido durante a qualificação do doutorado, cujos resultados foram publicados em 2018 na revista *Theoretical Computer Science*, veja em [32].

3.1 Relação entre árvores enraizadas e cografos

Nesta seção vamos identificar os cografos através de árvores enraizadas, onde introduziremos parâmetros nos vértices que permitirão escrever cada cografo numa forma única de árvore, que será chamada árvore ordenada.

Vimos que cada vértice interno de uma coárvore é *tipo*-1 ou *tipo*-0 de forma que em qualquer caminho $v_1, \ldots, v_k$ os tipos alternam ao longo do percurso, portanto o tipo dos vértices internos são determinados unicamente pelo tipo da raiz. Além disso, como a coárvore de um cografo é única a menos de permutações entre vértices de uma irmandade, então pela observação anterior cada árvore, cujos nós internos possuem ao menos dois filhos, representa exatamente duas coárvores: uma cuja raiz é *tipo*-0 e outra *tipo*-1, representando um cografo e seu respectivo complementar.

Definimos $\mathbb{T}$ como o conjunto das árvores enraizadas onde cada vértice interno possui ao menos dois filhos. Portanto cada elemento de $\mathbb{T}$ se refere à exatamente dois cografos distintos (conexo e desconexo).

Definição 3.1.1. *Seja $T \in \mathbb{T}$. Para cada vértice v de T definimos o* ***número de folhas***

***induzidas por v**, denotado por $l(v)$, como o número de folhas da árvore $T(v)$. No caso do próprio v ser uma folha, definimos $l(v) = 1$.*

O fato a seguir decorre diretamente da definição.

Fato 3.1.1. *Um vértice v de $T \in \mathbb{T}$ é uma folha se, e somente se, $l(v) = 1$.*

Exemplo 3.1.1. *Na árvore abaixo rotulamos cada vértice v com o valor $l(v)$.*

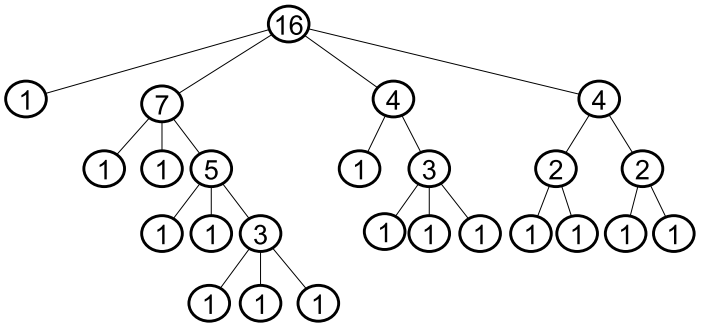

Figura 3.1: árvore rotulada

Fato 3.1.2. *Seja $T \in \mathbb{T}$, para cada v vértice interno de T vale $l(v) = \sum_{w \in filhos(v)} l(w)$.*

Demonstração. Seja $filhos(v) = \{v_1, \ldots, v_k\}$. Para cada $i \in \{1, \ldots, k\}$ as folhas da subárvore $T(v_i)$ são também folhas de de $T(v)$, portanto $l(v) \geq \sum_{w \in filhos(v)} l(w)$. Por outro lado qualquer folha de $T(v)$ é folha de alguma subárvore $T(v_i)$. Logo segue a igualdade. □

Como cada vértice interno de $T \in \mathbb{T}$ possui ao menos dois filhos, então pelo fato acima concluímos o seguinte resultado.

Fato 3.1.3. *Seja $T \in \mathbb{T}$, para cada vértice v de T, exceto a raiz, vale $l(v) < l(pai(v))$.*

Vale notar que dado um cografo G com n vértices, este é representado por uma árvore $T \in \mathbb{T}$, onde consequentemente $l(raiz(T)) = n$. Então é necessário particionar o conjunto $\mathbb{T}$ segundo o número de folhas induzidas na árvore, ou seja, para cada inteiro positivo n definimos o conjunto

$$\mathbb{T}_n = \{T \in \mathbb{T};\ l(raiz(T)) = n\}.$$

É fácil ver que $\bigcup_{n=1}^{\infty} \mathbb{T}_n$ é uma partição de $\mathbb{T}$, portanto a geração dos cografos com n vértices será feita através da geração dos elementos de $\mathbb{T}_n$, para cada n fixado.

Como estudaremos mais adiante algoritmos e complexidades, será útil termos conhecimento do número máximo de vértices em uma árvore de $\mathbb{T}_n$. O que é respondido pela proposição abaixo, já conhecida na literatura.

Proposição 3.1.1. *Seja $T \in \mathbb{T}_n$, o número de vértices de T é no máximo $2n-1$.*

Demonstração. Como $T \in \mathbb{T}_n$ então a árvore possui n folhas, provaremos por indução que o número de vértices internos é no máximo $n-1$.

Como cada nó interno possui no mínimo dois filhos, então a base da nossa indução é $n = 2$. Que por sua vez satisfaz o resultado, pois trata-se de uma árvore cuja raiz possui dois filhos folhas e portanto o número de nós internos é 1.

Suponha que para todo $T \in \mathbb{T}_n$, o número de nós internos de T é no máximo $n-1$.

Dado $T \in \mathbb{T}_{n+1}$, existe nó interno v cujos filhos são todos folhas, pois caso contrário a árvore não seria finita. Temos dois casos a considerar:

- CASO 1: Se v possui dois filhos, então considere a árvore T' obtida de T onde substituímos v por uma folha. Ou seja, para construção de T' retiramos um nó interno e duas folhas, seguida pela inserção de uma nova folha, portanto seu número folhas é $((n+1)-2+1) = n$. Então, pela hipótese de indução, tal árvore possui no máximo $n-1$ nós internos. Logo a árvore original T possui no máximo $(n-1+1)$ nós internos, donde segue o resultado neste caso.

- CASO 2: Se v possui três ou mais filhos, então considere a árvore T' obtida de T onde retiramos um único filho de v, portanto T' possui n folhas e, pela hipótese de indução, no máximo $n-1$ nós internos. Como o número de nós internos de T e T' são iguais, então a árvore T possui no máximo $(n-1)$ nós internos, o que garante o resultado, pois $n-1 < n$. O que finaliza a prova.

□

Como nosso interesse é gerar os elementos de $\mathbb{T}_n$, inicialmente vamos estudar as formas que as n folhas de uma árvore podem estar distribuídas, o que é dado pela possíveis formas de obter o valor $l(x)$ através de soma de inteiros positivos, que serão relacionadas às subárvores induzidas por seus filhos. Este estudo é chamado de *partição de inteiros*, que foi introduzido por Euler e desde então muita teoria foi desenvolvida. Usaremos aqui uma forma adaptada da definição, onde consideraremos que a partição deve possuir ao menos dois elementos, conforme formalizado abaixo.

Definição 3.1.2. *Seja $n \geq 2$ um inteiro positivo. Uma **partição** de n é uma sequência não-decrescente de números inteiros positivos $(a_i)_k := (a_1, a_2, a_3, \ldots, a_k)$, tal que $k \geq 2$ e $\sum_{i=1}^{k} a_i = n$. Denotaremos o conjuntos de todas as possíveis partições de n por $Part(n)$*

Exemplo 3.1.2. *$a = (1, 2, 5, 5)$ é uma partição do $n = 13$.*

Usaremos a ordenação *lexicográfica* para estabelecer uma ordenação total dentre partições de um inteiro n, conforme descrito abaixo.

Definição 3.1.3. *Dados $a, b \in Part(n)$, onde $a = (a_i)_k$ e $b = (b_i)_m$, a ordenação entre tais partições é dada pela ordenação lexicográfica. Isto é, tome $j = min\{i;\ a_i \neq b_i \text{ e } 1 \leq i \leq min\{k, m\}\}$, caso exista, e defina a relação de ordem entre a e b pela relação entre os inteiros a_j e b_j, se j não existir significa que $a = b$.*

Fato 3.1.4. *(**Tricotomia**) Vale a tricotomia de ordem em $Part(n)$, ou seja, dados $a, b \in Part(n)$ vale exatamente um dos seguintes casos: $(a < b)$ ou $(a = b)$ ou $(a > b)$.*

Segue uma tabela onde enumeramos todos os elementos de $Part(n)$ em ordem crescente, para $1 \leq n \leq 8$

n=2	n=3	n=4	n=5	n=6	n=7	n=8
1,1	1,1,1	1,1,1,1	1,1,1,1,1	1,1,1,1,1,1	1,1,1,1,1,1,1	1,1,1,1,1,1,1,1
	1,2	1,1,2	1,1,1,2	1,1,1,1,2	1,1,1,1,1,2	1,1,1,1,1,1,2
		1,3	1,1,3	1,1,1,3	1,1,1,1,3	1,1,1,1,1,3
		2,2	1,2,2	1,1,2,2	1,1,1,2,2	1,1,1,1,2,2
			1,4	1,1,4	1,1,1,4	1,1,1,1,4
			2,3	1,2,3	1,1,2,3	1,1,1,2,3
				1,5	1,1,5	1,1,1,5
				2,2,2	1,2,2,2	1,1,2,2,2
				2,4	1,2,4	1,1,2,4
				3,3	1,3,3	1,1,3,3
					1,6	1,1,6
					2,2,3	1,2,2,3
					2,5	1,2,5
					3,4	1,3,4
						1,7
						2,2,2,2
						2,2,4
						2,3,3
						2,6
						3,5
						4,4

Tabela 3.1: Partições de $n = 2, \ldots, 8$

Fato 3.1.5. *O menor elemento de Part(n) é* $a = (1, \ldots, 1)$ *e o maior é* $b = \left(\lfloor \frac{n}{2} \rfloor, \lceil \frac{n}{2} \rceil\right)$.

Considere um nó v de uma árvore $T \in \mathbb{T}$, onde $l(v) = 5$. A menos de uma permutação dentre os vértices, as únicas possíveis configurações para os filhos de v são:

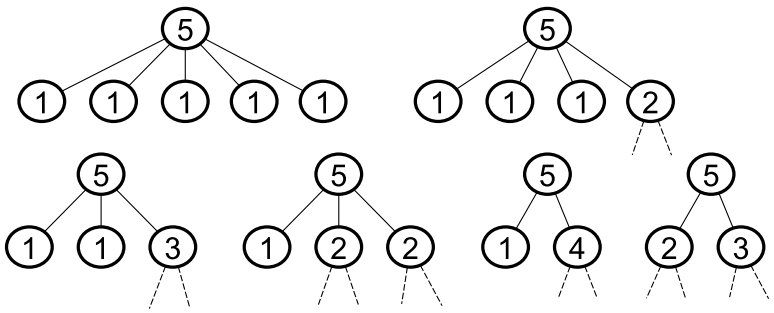

Figura 3.2: Configurações dos filhos de um vértice com 5 folhas induzidas

A definição abaixo relaciona partição de inteiro com distribuições de folhas numa

árvore.

Definição 3.1.4. *Seja $T \in \mathbb{T}$ e v um vértice interno de T. Escrevendo os filhos de v na ordem $v_1, \ldots, v_k$ de forma que $l(v_1), l(v_2), \ldots, l(v_k)$ forme uma sequência não-decrescente, a sequência $(l(v_1), \ldots, l(v_k))$ é chamada* ***partição induzida*** *por v.*

A partição induzida de um vértice está bem definida pelo fato 3.1.2.

Definição 3.1.5. *Uma árvore de $\mathbb{T}_n$ é dita* ***rotulada*** *se é conhecido o valor $l(v)$ para cada vértice v da árvore.*

O algoritmo abaixo faz uso do fato 3.1.2 para calcular o número de folhas induzidas em cada vértice de uma árvore.

Algorithm 3.1 Rotulação de árvore

Input: Árvore T em $\mathbb{T}$

Output: Árvore T onde cada vértice v possui o rótulo $l(v)$

```
1: procedure Rotular(T)
2:     for i ← nivel(T) to 1 do
3:         for all v ∈ nivel(T, i) do
4:             if v é folha then l(v) ← 1
5:             else l(v) ← Σ_{x∈F} l(x), onde F = filhos(v)
6:         end for
7:     end for
8: end function
```

A proposição 3.1.1 assegura o seguinte fato.

Fato 3.1.6. *Se $T \in \mathbb{T}_n$, a complexidade do procedimento Rotular(T) é $O(n)$.*

3.2 Ordenação de vértices e árvores

Como cada cografo está associado a uma única árvore de $\mathbb{T}$, a menos de uma permutação de vértices dentro de uma irmandade, é fundamental que seja estabelecida uma forma "padrão" de escrever coárvores isomorfas. Para isso estabeleceremos uma relação de ordem entre vértices e tomaremos como forma padrão aquela que possui todas as irmandades na forma que chamaremos "ordenadas". Dados vértices v e w é natural que a relação entre

estes seja estabelecida através dos valores $l(v)$ e $l(w)$, mas a questão é decidir qual critério de "desempate" deve ser tomado quando $l(v) = l(w)$?

Na árvore abaixo, cada irmandade está disposta em ordem não-decrescente segundo o número de folhas induzidas.

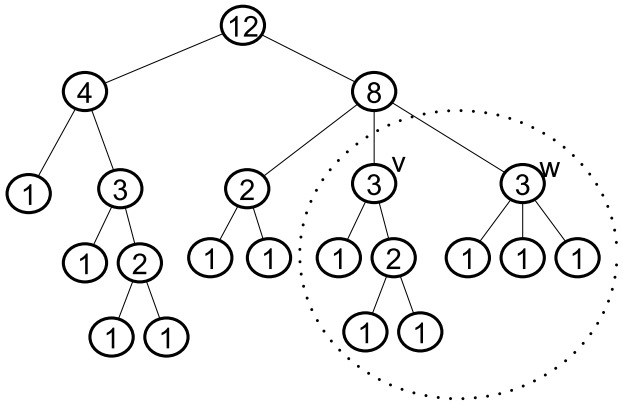

Figura 3.3: Árvore antes da ordenação.

Mas observe que na área indicada, existem dois vértices v e w tais que $l(v) = l(w) = 3$. Como uma permutação entre v e w geram cografos isomorfos, então precisamos definir uma ordem entre estes vértices, que será dada pela partição induzida por v e w: como v induz a partição $a = (1, 2)$ e w induz $b = (1, 1, 1) < a$, então diremos $w < v$ e podemos reescrever a árvore, conforme ilustrado abaixo.

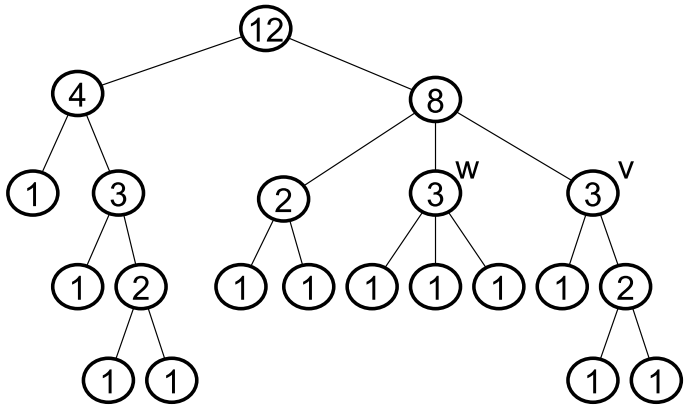

Figura 3.4: Árvore após ordenação.

A solução exibida acima estabelece um critério de desempate entre um par de vértices com mesma quantidade de folhas induzidas, mas o problema pode continuar quando as partições induzidas também empatam no nível abaixo. Então uma solução é aplicarmos o mesmo raciocínio nos níveis seguintes até que seja possível decidir uma ordem entre o

par de vértices. No pior dos casos haverá empate em todos os subníveis e diremos que os vértices são equivalentes, pois a permutação entre estes não altera em nada a configuração da árvore, isto é, geram árvores idênticas. A formalização desta ordenação total é dada pela definição abaixo.

Definição 3.2.1. *Sejam $T \in \mathbb{T}$ e v, w vértices de T. Estabelecemos as relações entre v e w.*

- *CASO 1: Se $l(v) < l(w)$ definimos $v < w$*
- *CASO 2: Se $l(v) > l(w)$ definimos $v > w$*
- *CASO 3: Se $l(v) = l(w) = 1$ definimos $v \backsimeq w$*
- *CASO 4: Se $l(v) = l(w)$, $l(v) \neq 1$:*

  Tome $(a_i)_k$ a partição induzida por v e $filhos(v) = \{v_1, \ldots, v_k\}$. Analogamente $(b_i)_m$ por w, onde $filhos(w) = \{w_1, \ldots, w_m\}$. Temos três subcasos a considerar:

  CASO 4.1: Se $(a_i)_k < (b_i)_m$ definimos $v < w$

  CASO 4.2: Se $(a_i)_k > (b_i)_m$ definimos $v > w$

  CASO 4.3: Se $(a_i)_k = (b_i)_m$, consequentemente $k = m$

  CASO 4.3a: Se $v_i = w_i$ para todo $1 \leq i \leq k$ definimos $v \backsimeq w$.

  CASO 4.3b: Senão tome $j = min\{i; v_i \neq w_i\}$. Se $v_j < w_j$ definimos $v < w$, senão $v > w$.

*No caso $v \backsimeq w$ diremos que v e w são **equivalentes**. E ao referir que $v \backsimeq w$ ou $v < w$ escrevemos simplesmente $v \leq w$, analogamente $v \geq w$.*

Vale observar que a definição pode ser estendida para vértices de árvores distintas, basta considerar uma terceira árvore onde os filhos da raiz são os vértices a serem comparados.

A tricotomia na ordenação de vértices é assegurada pela tricotomia de partições de inteiro, fato 3.1.4.

Fato 3.2.1. *(**Tricotomia**) Seja $T \in \mathbb{T}$ e considere v, w vértices de T, então vale exatamente um dos casos: $v < w$ **ou** $v = w$ **ou** $v > w$.*

No decorrer do texto ao usarmos o símbolo de igualdade entre vértices estaremos nos referindo, de fato, ao mesmo vértice. Por outro lado a definição 3.2.1 fornece a ideia intuitiva de que dois vértices são equivalentes quando as respectivas subárvores induzidas são idênticas, o que é garantimos pelo lema desenvolvemos a seguir.

Lema 3.2.1. *Sejam $T \in \mathbb{T}$ e v, w vértices de T, então $T(v) \cong T(w)$ se, e somente se, $v \backsimeq w$.*

Demonstração. A demonstração é trivial para o caso em que v é uma folha. Suponha que é v vértice interno de T e considere $(a_i)_k$ a sua partição induzida, onde $filhos(v) = \{v_1, \ldots, v_k\}$ e $a_i = l(v_i)$ para $i \in \{1, \ldots, k\}$. Analogamente tome $(b_i)_m$ induzida por w e $filhos(w) = \{w_1, \ldots, w_m\}$.

Suponha que $T(v) \cong T(w)$, então $k = l$ e $(a_i)_k = (b_i)_k$, onde vale ainda $\sum a_i = \sum b_i = l(v) = l(w) = N$. Vamos provar por indução sobre N que $v \backsimeq w$.

O resultado vale para o caso base $N = 2$, pois $Part(2) = \{(1, 1)\}$. Suponha que o resultado vale para valores menores que algum N. Como $T(v) \cong T(w)$ então $T(v_i) \cong T(w_i)$, para $i \in \{1, \ldots, k\}$, então $l(v_i) = l(w_i) < N$ e pela hipótese indutiva concluímos $v_i \backsimeq w_i$. Logo $v \backsimeq w$.

Reciprocamente suponha que $v \backsimeq w$, então $N = l(v) = l(w)$. Vamos novamente provar por indução sobre N que $T(v) \cong T(w)$. O resultado vale para o caso base onde $N = 2$ e suponha que vale para valores menores que algum N. Então pela hipótese e tricotomia, temos por definição que $v_i \backsimeq w_i$ para todo $i \in \{1, \ldots, k = m\}$, mas $l(v_i) = l(w_i) < N$ então pela hipótese indutiva vale $T(v_i) \cong T(w_i)$, logo $T(v) \cong T(w)$. □

Definição 3.2.2. *Sejam $T \in \mathbb{T}$ e v vértice de T. A irmandade $I_T(v) = \{v_1, \ldots, v_k\}$ é dita* ***ordenada*** *se $v_1 \leq \ldots \leq v_k$. Se todas as irmandades de uma árvore estão ordenadas, diremos que a árvore está ordenada.*

Exemplo 3.2.1. *A árvore representada abaixo está ordenada.*

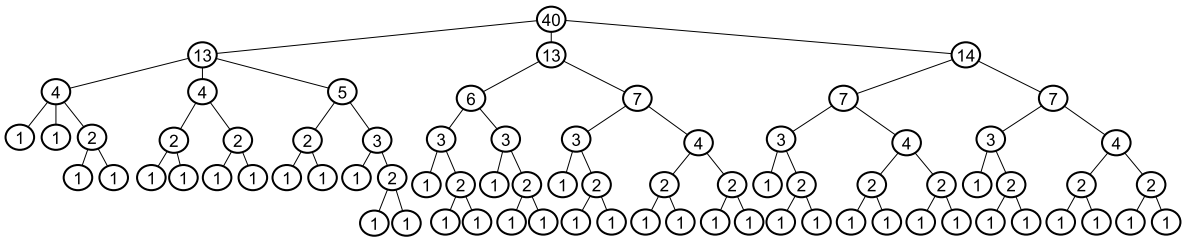

Figura 3.5: Árvore ordenada

O conceito de árvore ordenada estabelece uma forma padrão para a escrita da coárvore de um cografo, conforme provamos na proposição a seguir.

Proposição 3.2.1. *Cada cografo está associado a uma única árvore ordenada em $\mathbb{T}$.*

Demonstração. Seja G um cografo e tome T sua coárvore associada, em [12] é provado que T é única a menos de isomorfismos. Considere T' e T'' árvore ordenadas e ambas isomorfas a T. Como os vértices de T' e T'' possuem uma ordem de disposição dos vértices, então são árvores iguais a menos de permutação de vértices equivalentes de uma mesma irmandade. Porém o lema 3.2.1 assegura que essa permutação geram subárvores idênticas, com exatamente a mesma disposição de vértices. Logo T' e T'' são iguais. □

E então introduzimos uma ordenação total para as árvores de $\mathbb{T}_n$.

Definição 3.2.3. *Sejam T_1 e T_2 árvores em $\mathbb{T}$. Definimos $T_1 < T_2$ quando $raiz(T_1) < raiz(T_2)$ e $T_1 \backsimeq T_2$ no caso de equivalência entre as raízes.*

O lema 3.2.1 assegura que a definição acima é bem construída.

Fato 3.2.2. *O menor elemento de $\mathbb{T}_n$ é a árvore cuja raiz induz a partição $(1, \ldots, 1) \in Part(n)$ (veja o fato 3.1.5). Tal árvore está associada ao cografo completo K_n e seu complementar.*

Exemplo 3.2.2. *A árvore abaixo é a menor de $\mathbb{T}_6$ e ao lado representamos os respectivos cografos associados.*

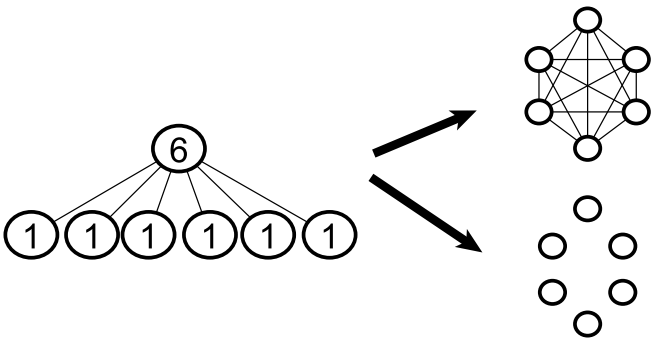

Figura 3.6: Menor árvore de $\mathbb{T}_6$ e respectivos cografos

3.2.1 Codificação de cografos

Nesta subseção usamos o conceito árvore ordenada para apresentar uma escrita única para cografos através de uma sequência de caracteres, chamada **codificação do cografo**.

Algorithm 3.2 Codificação

Input: Cografo G

Output: sequência de caracteres $cod(G)$

```
1:  procedure cod(G)
2:      Seja T(G) coárvore ordenada e rotulada
3:      Inicializa string S ← " "
4:      se raiz(T) é tipo-1 então S ← S + "C :"
5:      senão S ← S + "d :"
6:      S ← S + "l(raiz(T))/"
7:      pai ← raiz(T)
8:      for i ← 2 to nível(T) do
9:          for all v ∈ nivel(T, i) do
10:             if pai! = pai(v) then
11:                 S ← S + "*"
12:                 pai ← pai(v)
13:             end if
14:             S ← S + "l(v),"
15:         end for
16:         S ← S − ","                    ▷ Retira a vírgula excedente.
17:         S ← S + "/"
18:     end for
19:     return S
20: end function
```

Definição 3.2.4. *Seja G um cografo, a sequência de caracteres gerada pelo retorno algoritmo 3.2 é chamada codificação de G, e é denotada por $cod(G)$.*

Exemplo 3.2.3. *A codificação do cografo conexo cuja coárvore está tratada no exemplo 3.1.1 é:*

$$cod(G) = C/16/1, 4, 4, 7/1, 3{*}2, 2{*}1, 1, 5/1, 1, 1{*}1, 1{*}1, 1{*}1, 1, 3/1, 1, 1/$$

Onde a versão ordenada da árvore é representada a seguir:

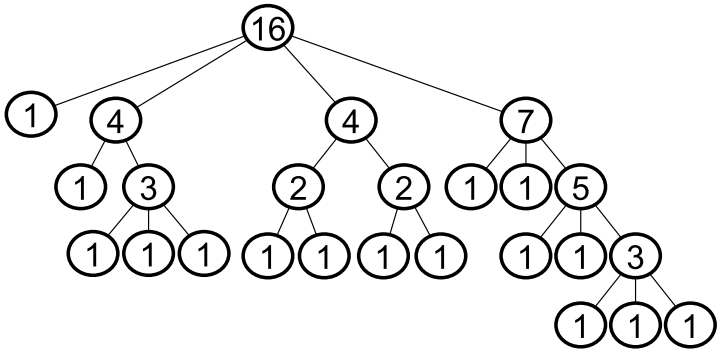

Figura 3.7: árvore do exemplo 3.1.1 ordenada

Fato 3.2.3. *Dado um cografo G e sua sequência $cod(G)$, vale observar que cada caractere:*

1. *"/" representa o fim ou início de um nível na árvore;*
2. *"$*$" simboliza o fim ou início de uma irmandade;*
3. *inteiro positivo representa um vértice da coárvore;*
4. *$'1'$ equivale a uma folha da árvore.*

O teorema abaixo garante que cada cografo possui uma única codificação, bem como cada codificação representa um único cografo.

Teorema 3.2.1. *Dois cografos G_1 e G_2, não triviais, são isomorfos se, e somente se, $cod(G_1) = cod(G_2)$.*

Demonstração. Sejam G_1 e G_2 cografos e tome suas respectivas coárvores T_1 e T_2 ordenadas.

($\Rightarrow$) Suponha que $G_1 \cong G_2$, então as coárvores T_1 e T_2 são idênticas, pois antes da ordenação a única distinção era por permutações de irmãos. Então o algoritmo 3.2 retorna $cod(G_1) = cod(G_2)$.

($\Leftarrow$) Faremos a prova por contra positiva. Ou seja, dados $G_1(V_1, E_1)$ e $G_2(V_2, E_2)$ cografos não isomorfos, vamos provar que $cod(G_1) \neq cod(G_2)$. Podemos supor que G_1 e G_2 têm a mesma quantidade de vértices, pois caso contrário o fato 3.2.3.4 garante que $cod(G_1) \neq cod(G_2)$, e a prova está finalizada. Por raciocínio semelhante também podemos supor que ambos os cografos possuem a mesma conexidade, isto é, ou são ambos conexos ou ambos desconexos. Então para provar a recíproca do teorema basta provar a implicação:

Se G_1 e G_2 são cografos não isomorfos, ambos com n vértices e mesma conexidade, então $cod(G_1) \neq cod(G_2)$

Para tal, faremos por indução sobre o número de vértices n. Sem perda de generalidade, consideraremos apenas o caso de grafos conexos.

O caso base se sustenta em $n = 2$, onde o resultado vale por vacuidade, já que neste caso existe somente um cografo conexo (e outro desconexo).

Suponha que se dois cografos são não isomorfos, com a mesma conexidade e a mesma quantidade k de vértices, onde $k \leq n$, então suas codificações são distintas.

Sejam G_1 e G_2 cografos não isomorfos, conexos e com $n+1$ vértices. Considere ainda T_1 e T_2 suas respectivas coárvores ordenadas, enraizadas em r_1 e r_2, nesta ordem. Então pelo lema 3.2.1 vale $r_1 \neq r_2$. Sem perda de generalidade podemos supor que $r_1 < r_2$.

Considere $filhos(r_1) = \{v_1, \ldots, v_k\}$ e $filhos(r_2) = \{w_1, \ldots, w_l\}$, tais que $v_1 \leq \ldots \leq v_k$ e $w_1 \leq \ldots \leq v_l$ e tome ainda, nesta ordem, as partições induzidas $(a_i)_k$ e $(b_i)_l$ por r_1 e r_2, onde $l(v_i) = a_i$ e $l(w_i) = b_i$. Então

$$\begin{cases} cod(G_1) = C : l(r_1)/a_1, a_2, \ldots, a_k/\ldots \\ cod(G_2) = C : l(r_2)/b_1, b_2, \ldots, b_l/\ldots \end{cases} . \tag{3.1}$$

Pela tricotomia, os vértices em questão satisfazem o caso 4.1 ou 4.3b da definição 3.2.1:

- **CASO 4.1:** $(a_i)_k < (b_i)_l$, então pela equação (3.1) temos $cod(G_1) \neq cod(G_2)$ e a prova está finalizada.
- **CASO 4.3b:** $k = l$ e $(a_i)_k = (b_i)_k$, onde $\exists j \in \{1, \ldots, k\}$ tal que $v_j < w_j$. Então, valem os seguintes fatos:
  - Como $v_j < w_j$, então o lema 3.2.1 garante que as coárvores $T_1(v_j)$ e $T_2(w_j)$ são não isomorfas.
  - Como $(a_i)_k = (b_i)_k$, então $l(v_j) = a_j = b_j = l(w_j)$, ou seja, $T_1(v_j)$ e $T_2(w_j)$ possuem cada $l(v_j)$ folhas.
  - Pelo fato 3.1.3 temos $l(v_j) < l(r_1) = n$.
  - Vale $l(w_j) = l(v_j) > 1$, pois caso contrário v_j e w_j seriam folhas e teríamos $v_j = w_j$, o que geraria contradição.

– Como r_1 e r_2 são *tipo*-1, então v_j e w_j são *tipo*-0.

Sejam G_v e G_w os cografos referentes às coárvores $T_1(v_j)$ e $T_2(w_j)$. As observações acima garantem que os cografos não são isomorfos, possuem a mesma quantidade de vértices (menor que n), não são grafos triviais e têm a mesma conexidade. Então pela hipótese indutiva vale $cod(G_v) \neq cod(G_w)$, o que garante que $cod(G_1)$ e $cod(G_2)$ são distintas.

□

3.3 Gerando árvores em ordem

Assegurado pela proposição 3.2.1, a geração dos cografos será feita através da geração dos elementos de $\mathbb{T}_n$, onde iniciamos na menor árvore do conjunto e em seguida geramos a árvore imediatamente maior que a antecessora, até alcançar a maior do conjunto. A construção deste procedimento será feita nesta seção, que iniciamos com a formalização da noção de elemento imediatamente maior.

Definição 3.3.1. *Seja $\mathbb{X}$ um conjunto não-vazio munido de ordem, onde vale a tricotomia. Dados $a, b \in \mathbb{X}$, diremos que o elemento b é imediatamente maior que a em $\mathbb{X}$ se $b \in \mathbb{X}$ e $b > a$, onde para qualquer $c \in \mathbb{X}$ tal que $c > a$ então $c \geq b$. Naturalmente se a não possui elemento imediatamente maior em $\mathbb{X}$, então a é o maior elemento do conjunto.*

Exemplo 3.3.1. *Pela tabela 3.1, vemos que a partição imediatamente maior que $a = (1, 1, 2, 2)$ em $Part(6)$ é $b = (1, 1, 4)$.*

Desenvolvemos a seguir o procedimento que obtém a partição imediatamente maior.

Algorithm 3.3 Partição imediatamente maior

Input: $(a_i)_k = (a_1, a_2, \ldots, a_k) \in Part(n)$

Output: $(b_i)_m$ imediatamente maior que $(a_i)_k$ em $Part(n)$

1: **procedure** *ParticaoSeguinte*$((a_i)_k)$
2: $n \leftarrow \sum_{i=1}^{k} a_i$
3: **if** $a_1 \neq \lfloor n/2 \rfloor$ **then**
4: **if** $a_k - a_{k-1} \leq 1$ **then return** $(a_1, a_2, \ldots, a_{k-2}, a_{k-1} + a_k)$;
5: **else** ▷ $a_k - a_{k-1} > 1$
6: $a_{k-1} \leftarrow a_{k-1} + 1$;
7: $a_k \leftarrow a_k - 1$;
8: Tome q e r o quociente e resto da divisão a_k por a_{k-1};
9: **if** $q > 1$ **then return** $(a_1, \ldots, a_{k-2}, \underbrace{a_{k-1}, \ldots, a_{k-1}}_{qvezes}, (a_{k-1} + r))$;
10: **else return** $(a_1, \ldots, a_{k-2}, a_{k-1}, a_k)$;
11: **end if**
12: **else**
13: **if** $n \neq 3$ **then return** ***null***
14: **else** ▷ $n = 3$ caso especial
15: **if** $a_2 = \lceil n/2 \rceil$ **then return** ***null*** ;
16: **else return** $(1, 2)$;
17: **end if**
18: **end if**
19: **end function**

Vamos provar a corretude do algoritmo acima.

Proposição 3.3.1. *Se $a \in Part(n)$, o retorno do algoritmo 3.3 é a partição imediatamente maior que* ***a*** *em $Part(n)$. Caso o retorno seja* ***null*** *então a é o maior elemento de $Part(n)$.*

Demonstração. Seja $a = (a_i)_k \in Part(n)$. Para o caso $n = 3$ temos apenas as partições $(1, 1, 1)$ e $(1, 2)$, garantindo a corretude do algoritmo.

Se $n = 2$, então $(1, 1)$ é a única partição, portanto é a maior do conjunto $Part(2)$. Neste caso $a_1 = \left\lfloor \frac{n}{2} \right\rfloor = 1$ e a linha 13 retornará ***null*** .

Pela definição de partição, necessariamente $a_1 \leq \left\lfloor \frac{n}{2} \right\rfloor$. Suponha que $n > 3$ e $a_1 < \left\lfloor \frac{n}{2} \right\rfloor$. Vamos separar a prova em casos:

CASO 1: $a_k - a_{k-1} \leq 1$.

Então $b := (b_i)_{k-1} = b_1, b_2, \ldots, b_{k-1}$, onde $\begin{cases} b_i = a_i, \text{ se } 1 \leq i \leq k-2 \\ b_{k-1} = a_{k-1} + a_k \end{cases}$,
é a partição retornada pelo algoritmo.

É claro que $b > a$ e que $(b_i)_{k-1} \in Part(n)$. Vamos provar que b é a sequência imediatamente maior que a.

Seja $c = \{c_i\}_m \in Part(n)$ tal que $c > a$. Queremos provar que $c \geq b$.

Por definição $\exists j \leq k$, sendo j o menor índice tal que $c_j > a_j$.

- Se $j \leq k-2$ então $c_j > a_j = b_j$, donde segue $c > b$ e a prova do corrente caso está finalizada.
- Se $j = k-1$ temos $\begin{cases} c_i = a_i = b_i, \text{ se } 1 \leq i \leq k-2 \quad (*) \\ c_{k-1} > a_{k-1} \end{cases}$

  Então $c_{k-1} \geq a_{k-1} + 1 \geq a_k$, pela hipótese do presente caso.

  Observe que c tem $k-1$ termos, pois caso contrário teríamos $c_k \geq c_{k-1} \geq a_k$, então $\sum^k c_i > \sum^k a_i = n$, o que é absurdo.

  Portanto,
  $$\sum_{i=1}^{k-2} c_i + c_{k-1} = \sum_{i=1}^{k-2} a_i + a_{k-1} + a_k \Rightarrow c_{k-1} = a_{k-1} + a_k,$$
  onde a última implicação é obtida das equações $(*)$. Logo $c_{k-1} = b_{k-1}$, ou seja, $c = b$.
- Por último, se $j = k$ então $\sum^k c_i > \sum^k a_i = n$, o que é absurdo.

Logo, vale $c \geq b$.

CASO 2:$a_k - a_{k-1} > 1$

Neste caso as linhas 6 e 7, geram a partição $b := (b_i)_k = b_1, b_2, \ldots, b_k$,
onde $\begin{cases} b_i = a_i, \text{ se } 1 \leq i \leq k-2 \\ b_{k-1} = a_{k-1} + 1 \\ b_k = a_k - 1 \end{cases}$

Observe que
$$\sum^k b_i = \sum^{k-2} b_i + b_{k-1} + b_k = \sum^{k-2} a_i + a_{k-1} + 1 + a_k - 1 = n,$$
então $b \in Part(n)$. Tome q e r o quociente e resto da divisão b_k por b_{k-1} (linha 8).

Temos dois subcasos a considerar:

- **CASO 2a:** $q > 1$

  O retorno do algoritmo é
  $b' = (b_1, \ldots, b_{k-2}, b_{k-1}, \underbrace{b_{k-1}, \ldots, b_{k-1}}_{q-1 \text{ vezes}}, (b_{k-1} + r))$, que possui $k + q - 1$ termos.

  Observe que $\begin{cases} b'_i = a_i, \text{ se } 1 \leq i \leq k-2 \\ b'_i = a_{k-1} + 1, \text{ se } k-1 \leq i \leq k+q-2 \\ b'_{k+q-1} = a_{k-1} + 1 + r \end{cases}$.

$$\begin{aligned}
\sum_{i=1}^{k+q-1} b'_i &= (\sum_{i=1}^{k-2} b'_i) + b'_{k-1} + (\sum_{i=k}^{k+q-2} b'_i) + b'_{k+q-1} \\
&= (\sum_{i=1}^{k-2} a_i) + (a_{k-1}+1) + \sum_{i=k}^{k+q-2} (a_{k-1}+1) + (a_{k-1}+1+r) \\
&= (\sum_{i=1}^{k-2} a_i) + (a_{k-1}+1) + (a_{k-1}+1).(q-1) + (a_{k-1}+1+r) \\
&= (\sum_{i=1}^{k-2} a_i) + (a_{k-1}+1) + (a_{k-1}+1)q + r \\
&= (\sum_{i=1}^{k-2} a_i) + (a_{k-1}+1) + [(b_{k-1})q + r] \\
&= (\sum_{i=1}^{k-2} a_i) + (a_{k-1}+1) + b_k \\
&= (\sum_{i=1}^{k-2} a_i) + (a_{k-1}+1) + a_k - 1 = \sum_{i=1}^{k} a_i = n.
\end{aligned}$$

  Portanto $b' \in Part(n)$. Pela construção de b' é fácil ver que $b' \geq a$.

  Vamos provar que b' é a partição imediatamente maior que a em $Part(n)$.

  Seja $c = (c_i)_m \in Part(n)$ tal que $c > a$. Vamos provar que $c \geq b'$.

  Por definição $\exists j \leq k$, sendo j o menor índice tal que $c_j > a_j$. De forma semelhante ao caso anterior, temos:

  ◦ Se $j \leq k-2$ então $c_j > a_j = b'_j$, donde segue que $c > b'$.

  ◦ Se $j = k-1$ temos $c_{k-1} > a_{k-1} \Rightarrow c_{k-1} \geq a_{k-1} + 1 = b'_{k-1}$. Como $c_i = a_i = b_i$ para $1 \leq i \leq k-2$, então $c \geq b'$.

  ◦ Se $j = k$ então $\sum^k c_i > \sum^k a_i = n$, o que é absurdo.

Logo vale $c \geq b'$

- **CASO 2b:** $q = 1$

  Neste caso o algoritmo retorna a própria partição $b = (b_i)_k$. Como $b_{k-1} = a_{k-1}+1 > a_{k-1}$ então $b > a$.

  Vamos provar que b é a sequência imediatamente maior que a. Seja $c = \{c_i\}_m \in Part(n)$ tal que $c > a$. Queremos provar que $c \geq b$. Por definição $\exists j \leq k$ o menor índice tal que $c_j > a_j$.

  De forma semelhante aos casos anteriores, temos:

  ◦ Se $j \leq k-2$ então $c_{k-2} > a_{k-2} = b_{k-2}$, donde segue que $c > b$.

  ◦ Se $j = k-1$ temos $c_{k-1} > a_{k-1} \Rightarrow c_{k-1} \geq a_{k-1} + 1 = b_{k-1}$. Como $c_i = a_i = b_i$ para $1 \leq i \leq k-2$, então $c \geq b$

  ◦ Se $j = k$ então $\sum^k c_i > \sum^k a_i = n$, o que é absurdo. Logo vale $c \geq b'$

Por fim, suponha que $n > 3$ e $a_1 = \lfloor \frac{n}{2} \rfloor$. Então pelo fato 3.1.5, $(a_i)_k$ é a maior partição de n, o que é de acordo com o algoritmo que retornará "***null*** "na linha 13. □

Proposição 3.3.2. *O algoritmo ParticaoSeguinte*$((a_i)_k)$ *tem complexidade* $O(n)$ *em tempo de execução, onde* $a_k \in Part(n)$.

Demonstração. Considere uma partição $a = (a_i)_k \in Part(n)$ e $q = \left\lfloor \frac{a_k - 1}{a_{k-1}+1} \right\rfloor$, o algoritmo pode ser dividido em casos.

- **CASO 1:** $a_1 \neq \lfloor n/2 \rfloor$

  CASO 1a: $a_k - a_{k-1} \leq 1$ (linha 4)

  CASO 1b: $a_k - a_{k-1} > 1$ e $q > 1$ (linha 9)

  CASO 1c: $a_k - a_{k-1} > 1$ e $q < 1$ (linha 10)
- **CASO 2:** $a_1 = \lfloor n/2 \rfloor$ (linha 12)

É claro que os casos 1a ,1c e 2 são executados em tempo constante, enquanto o caso 1b é efetuado com $O(q)$ operações, cujo pior ocorre ao maximizar q, isto é, quando a entrada do procedimento é $(a_i)_2 = (1, n-1)$, onde obtemos $q = \frac{n-2}{2}$. Logo a complexidade de pior caso do algoritmo é $O(n)$. □

Definição 3.3.2. *Seja* $T \in \mathbb{T}$ *uma árvore ordenada, um vértice* v *de* T *é dito* ***esgotado*** *se é uma folha ou sua partição induzida é o maior elemento de* $Part(l(v))$.

O algoritmo abaixo faz uso direto da definição para verificar se um vértice está esgotado.

Algorithm 3.4 Vértice esgotado

Input: árvore $T \in \mathbb{T}$ ordenada e rotulada e vértice v de T.

Output: valor *booleano*

```
1: procedure Esgotado(T, v)
2:     if l(v) > 1 then
3:         Seja filhos(v) = {v_1, v_2, ..., v_k} ordenado.
4:         if l(v_1) = ⌊l(v)/2⌋ and l(v_2) = ⌈l(v)/2⌉ then return true
5:         else return false
6:         end if
7:     else return true
8:     end if
9: end function
```

Fato 3.3.1. *O algoritmo Esgotado(T, v) possui complexidade $O(1)$ em tempo de execução.*

A seguir definiremos o vértice pivô de uma árvore $T \in \mathbb{T}_n$, cuja finalidade é indicar como gerar a árvore T' imediatamente maior que T em $\mathbb{T}_n$.

Um percurso numa árvore é uma "visita" sistemática a cada um de seus vértices, onde entende-se por "visita" alguma operação específica sobre o vértice, por exemplo sua impressão ou consulta de dados. O percurso que trataremos aqui no texto visita cada vértice uma única vez.

Dada uma árvore ordenada $T \in \mathbb{T}$ definimos o percurso **pós-ordem invertido** através do seguinte procedimento recursivo: primeiro são visitados as subárvores referentes aos seus filhos da direita para a esquerda em pós-ordem invertido e por último é feita a visitação de $raiz(T)$, conforme descrito no procedimento a seguir:

Algorithm 3.5 Percurso em Pós-ordem Invertida

Input: árvore $T \in \mathbb{T}$ ordenada e vértice v

Output: sequência de vértices $v_1, \ldots, v_m$

```
1: procedure PosOrdemInvertida(T, v)
2:     if v não é folha then
3:         Seja filhos(v) = {v_1, v_2, ..., v_k}
4:         for i ← k to 1 do PosOrdemInvertida(T, v_i)
5:     end if
6:     visita(v)
7: end function
```

Exemplo 3.3.2. *A impressão dos vértices da árvore abaixo por um percurso pós-ordem invertida é "$k, j, i, h, g, d, f, e, c, b, a$".*

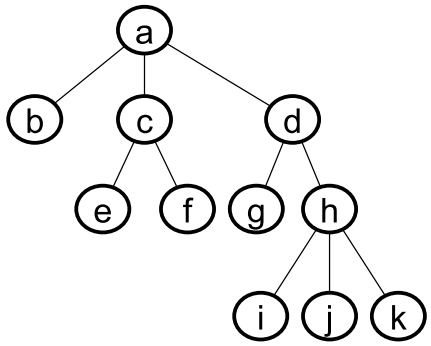

Figura 3.8: Percurso em pós-ordem invertida

Como cada vértice do percurso pós-ordem invertida é visitado uma única vez, então a proposição 3.1.1 assegura a complexidade do procedimento, enunciada abaixo.

Fato 3.3.2. *Sejam $T \in \mathbb{T}_n$ ordenada e rotulada e v um vértice de T, então $PosOrdemInvertida(T, v)$ tem complexidade $O(l(v)))$.*

Em particular, ao fazer a chamada $PosOrdemInvertida(T, raiz(T))$ para $T \in \mathbb{T}_n$, será feito o percurso em toda a árvore, cuja complexidade é $O(l(raiz(T))) = O(n)$.

Definição 3.3.3. *Seja $T \in \mathbb{T}$ ordenada. Suponha que a ordem de visitação do percurso pós-ordem invertida em T é $v_1, \ldots, v_m$. O pivô de T, caso exista, é o vértice v_i de menor índice tal que v_i não é esgotado.*

Exemplo 3.3.3. *O vértice em destaque é o pivô da árvore.*

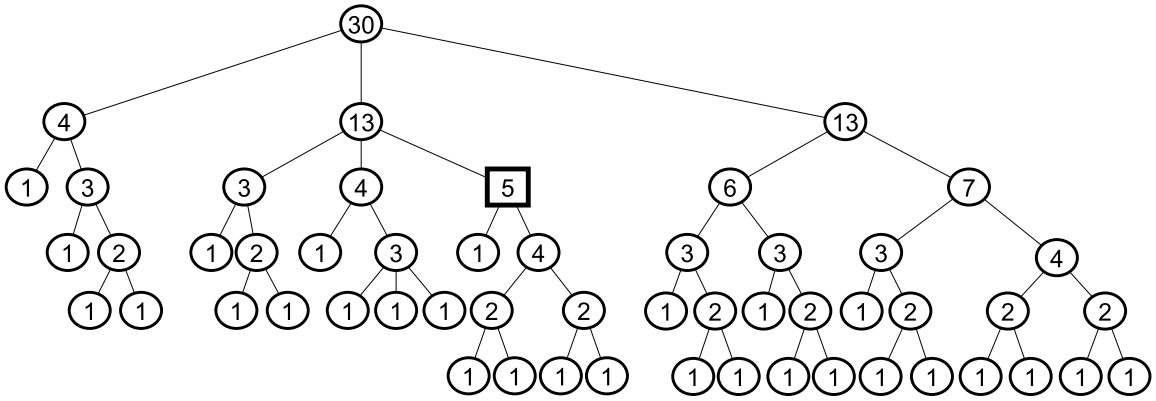

Figura 3.9: Pivô de uma árvore

O algoritmo abaixo é capaz de determinar o pivô de uma árvore $T \in \mathbb{T}$, caso não exista retornará ***null*** .

Algorithm 3.6 Pivô de uma árvore

Input: árvore $T \in \mathbb{T}_n$ ordenada e rotulada

Output: vértice v pivô de T

```
1:  procedure EncontraPivo(T, v)
2:      if v não é folha then
3:          Seja filhos(v) = {v_1, v_2, ..., v_k} ordenado.
4:          for i ← k to 1 do
5:              w ← EncontraPivo(T, v_i)
6:              if w ≠ null then return w
7:              end if
8:          end for
9:          if Esgotado(T, v) then return null
10:         else return v
11:         end if
12:     else return null
13:     end if
14: end function
```

Proposição 3.3.3. *Seja $T \in \mathbb{T}$. O vértice retorno de $EncontraPivo(T, raiz(T))$ é o pivô de T, caso o retorno seja **null** então a árvore não possui pivô.*

Demonstração. O procedimento faz um percurso em pós-ordem invertida e retorna o primeiro vértice não esgotado encontrado. Caso tal vértice não exista o retorno de todas

as chamadas recursivas será ***null*** . □

A complexidade decorre diretamente do fato 3.3.1 e descrevemos abaixo.

Fato 3.3.3. *Sejam $T \in \mathbb{T}_n$ ordenada e rotulada e v vértice de T, então a complexidade de $EncontraPivo(T, v)$ é $O(l(v))$.*

Em particular o custo para buscar o pivô em uma árvore $T \in \mathbb{T}_n$ é $O(n)$, pela chamada $EncontraPivo(T, raiz(T))$.

Escrevemos a seguir um algoritmo que reconstrói um vértice a partir de uma partição dada como entrada, sem alterar o número de folhas da árvore. Tal algoritmo auxiliará na demonstração do lema 3.3.1 e na construção do algoritmo 3.8.

Algorithm 3.7 Reconstrução de vértice através de partição

Input: vértice v de árvore $T \in \mathbb{T}_n$ ordenada e partição $(a_i)_k \in Part(l(v))$

Output: vértice v' com partição induzida $(a_i)_k$

```
1: procedure ReconstroiVertice(v, (a_i)_k)
2:     Substitua os filhos de v pelos vértices v_1 ..., v_k
3:     for i ← 1 to k do
4:         if a_i > 1 then insira a_i folhas em v_i
5:     end for
6:     return v
7: end function
```

Exemplo 3.3.4. *Abaixo representamos as árvores $T, T' \in \mathbb{T}_{12}$, nesta ordem. Onde a árvore T' é obtida por T ao substituir a subárvore enraizada em v por v', dado pelo retorno de $ReconstroiVertice(v, (a_i)_7)$, onde $(a_i)_7 = (1, 6)$.*

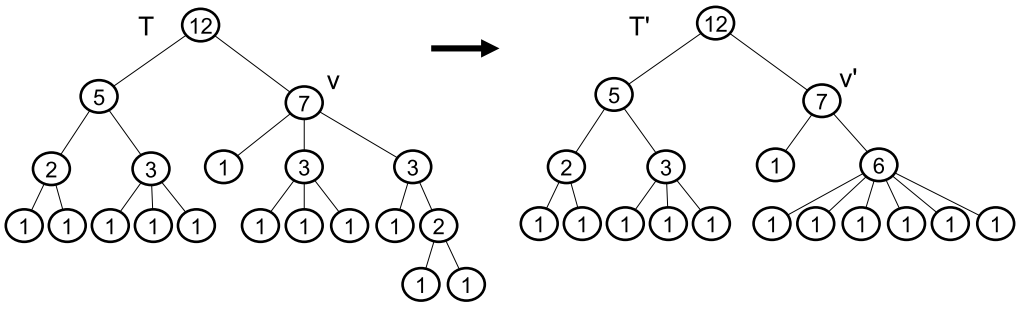

Figura 3.10: Reconstrução de vértice.

Fato 3.3.4. *Sejam $T \in \mathbb{T}_n$ ordenada, v um vértice e $(a_i)_k \in Part(l(v))$, então o procedimento $ReconstroiVertice(v, (a_i)_k)$ tem complexidade $O(l(v))$.*

Demonstração. Denotaremos os filhos de v antes da substituição por $u_1 \leq \ldots \leq u_m$ e após a substituição por $v_1 \leq \ldots \leq v_k$, pela linha 2. O tempo para substituição deste processo é:

$$\sum_1^m O(l(u_i)) + \sum_1^k O(l(v_k)) = O(l(v)) + O(l(v)) = O(l(v))$$

Onde a primeira igualdade decorre do fato 3.1.2. Com raciocínio semelhante vemos que o *loop* da linha 3 é executado em tempo $O(l(v))$, logo segue a complexidade desejada. □

Lema 3.3.1. *Uma árvore é o maior elemento de $\mathbb{T}_n$ se, e somente se, não possui pivô.*

Demonstração. Faremos a prova por contra-positiva.

Seja $T \in \mathbb{T}_n$ com pivô v, cuja partição induzida é $\mathbf{a} \in Part(l(v))$, portanto existe $\mathbf{b} \in Part(l(v))$ tal que $b > a$. Considere a árvore T' obtida de T após substituição de v pelo vértice v' dado pelo retorno de $ReconstroiVertice(v, \mathbf{b})$. Então $T' > T$, ou seja, T não é o maior elemento de $\mathbb{T}_n$.

Reciprocamente, suponha que T não é o maior elemento de $\mathbb{T}_n$, então existe algum vértice que não está esgotado, portanto o pivô da árvore existe. □

Exemplo 3.3.5. *As árvores abaixo são a menor e a maior árvores de $\mathbb{T}_{15}$, nesta ordem.*

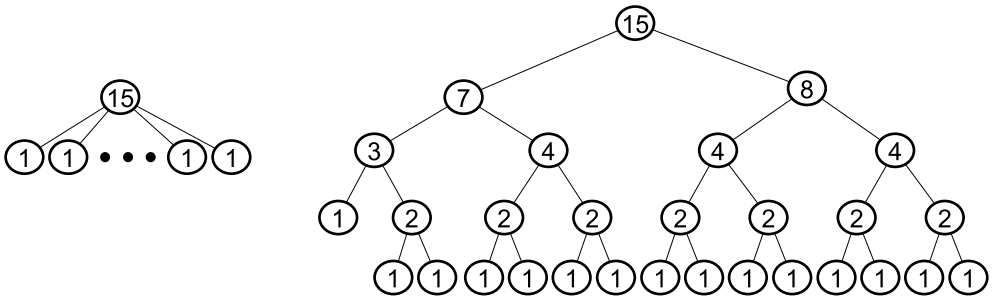

Figura 3.11: Menor e maior árvore de $\mathbb{T}_{15}$

Observe que a maior árvore de $\mathbb{T}_n$ é binária, para $n \geq 2$. De fato, o lema 3.3.1 garante que todos os vértices são esgotados, então a partição induzida por cada vértice possui apenas dois elementos, pelo fato 3.1.5. Porém a recíproca não é verdadeira, como observado no exemplo abaixo.

Exemplo 3.3.6. *A árvore abaixo é binária, porém não é a maior de $\mathbb{T}_{15}$, pois o vértice em destaque é seu pivô.*

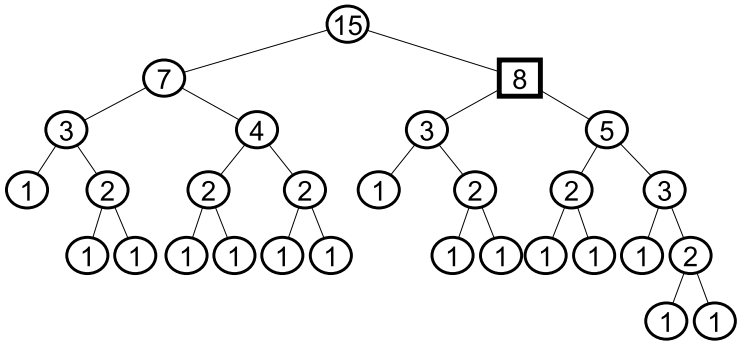

Figura 3.12: árvore binária e não maior de $\mathbb{T}_{15}$

Agora somos capazes de desenvolver o procedimento que obtém a árvore imediatamente maior, que fará a construção de cada árvore entre a menor e maior de $\mathbb{T}_n$ (exemplo 3.3.5). Tal procedimento é tão importante quanto o algoritmo gerador de cografos, que será construído mais adiante.

A fim de não sobrecarregar a escrita do algoritmo fixaremos a seguinte notação: Dada a irmandade $I(v_j) = \{v_1, \ldots, v_{j-1}, v_j, v_{j+1}, \ldots, v_m\}$ ordenada, definimos os conjuntos $I^-(v_j) = \{v_1, \ldots, v_{j-1}\}$ e $I^+(v_j) = \{v_{j+1}, \ldots, v_m\}$, então $I(v_j)$ pode ser particionado em $I^-(v_j) \cup \{v_j\} \cup I^+(v_j)$.

Algorithm 3.8 Árvore imediatamente maior

Input: árvore $T \in \mathbb{T}_n$ ordenada e rotulada

Output: árvore $T' \in \mathbb{T}_n$ imediatamente maior que T

1: **procedure** *ArvoreSeguinte*(T)
2: $v \leftarrow EncontraPivo(T, raiz(T))$
3: **if** $v \neq$ ***null*** **then**
4: $b_m \leftarrow ParticaoSeguinte((a_i)_k)$, onde $(a_i)_k$ é partição induzida de v.
5: $v \leftarrow ReconstroiVertice(v, (b_i)_m)$
6: $x \leftarrow v$
7: **repeat**
8: **for all** $y \in I^+(x)$ **do**
9: **if** $l(y) = l(x)$ **then** copiar subárvore $T(x)$ em $T(y)$
10: **else** $y \leftarrow ReconstroiVertice(y, c)$, onde $c = (1)_{l(y)}$
11: **end for**
12: $x \leftarrow pai(x)$
13: **until** $x =$ ***null***
14: **return** T
15: **else return** ***null*** ▷ não existe árvore maior
16: **end if**
17: **end function**

A ideia geral do algoritmo acima é buscar o pivô da árvore, e em seguida evoluir sua partição induzida e "*resetar*" para a menor configuração possível todos os vértices que a busca do pivô visitou anteriormente. Essa ideia é baseada no fato de que a comparação de árvores é feita da esquerda para a direita em cada irmandade (definição 3.2.1), enquanto a busca do pivô ocorre da direita para esquerda.

Observe que o algoritmo 3.8 exige que a árvore de entrada seja ordenada, contudo uma propriedade interessante é que a árvore retornada mantém a ordenação. Conforme descrito e provado na proposição a seguir.

Proposição 3.3.4. *Seja $T \in \mathbb{T}_n$ ordenada tal que T não é o maior elemento de $\mathbb{T}_n$, então a árvore referente ao retorno de $ArvoreSeguinte(T)$ está ordenada.*

Demonstração. Seja T' o retorno do procedimento, observe que $T' \in \mathbb{T}_n$, pois o número de folhas em cada vértice da árvore é mantido. Denotaremos por v o pivô de T, cuja existência é garantida pelo lema 3.3.1, e por v' o correspondente de v em T', pela operação da linha

5. Em geral para cada x em T denotaremos por x' o seu correspondente em T', caso exista, segundo as operações nas linhas 9 e 10.

Para cada x' ancestral de v', provaremos que as irmandades $I_{T'}(x')$ e $I_{T'}(v')$ estão ordenadas. Pois as demais irmandades não sofreram alteração, então a ordenação decorre diretamente da ordenação de T. Mais que isso, pelo mesmo raciocínio vemos que o conjunto $I^-_{T'}(x')$ já está ordenado para cada x', restando apenas provar a ordenação de $I^+_{T'}(x')$ e $I^+_{T'}(v')$. De fato para cada $y' \in I^+_{T'}(x') \cup \{x'\}$ valem os casos:

1. Se $x \leq y$ e $l(x) = l(y)$ então pela operação da linha 9 e lema 3.2.1 vale $x' \backsimeq y'$.
2. Se $x < y$ e $l(x) < l(y)$ então a partição induzida por y' é a menor de $Part(l(y))$ (linha 10), então $x' < y'$, segundo a definição 3.2.1.

Portanto em ambos os casos a ordenação dos vértices é mantida, isto é $I_{T'}(x')$ está ordenada. Resta provar a ordenação de $I_{T'}(v')$.

Escrevemos $I_T(v) = \{v_1 \leq \ldots \leq v_j \leq v_{j+1} \leq \ldots \leq v_{j+h}\}$ e $I_{T'}(v') = \{v'_1 \leq \ldots \leq v'_j \leq v'_{j+1} \leq \ldots \leq v'_{j+h}\}$, onde $v_j := v$ e $v'_j := v'$.

Como o algoritmo só altera os vértices do conjunto $I^+_T(v) \cup \{v\}$, então a ordenação de T garante a ordenação de $I^-_{T'}(v')$.

Afirmação: v' está ordenado em relação ao conjunto $I_{T'}(v')$.

Precisamos provar que $v'_{j-1} \leq v'_j \leq v'_{j+1}$. De fato, pela proposição 3.3.1 e caso 4.1 da definição 3.2.1 temos $v'_j > v_j$, mas pela ordenação de T vale $v_j \geq v_{j-1} \backsimeq v'_{j-1}$, portanto $v'_j \geq v'_{j-1}$. Por outro lado se $l(v_{j+1}) = l(v_j)$ então pela linha 9 temos $v'_{j+1} \backsimeq v'_j$, senão $l(v_{j+1}) > l(v_j)$ (pela linha 10 e definição 3.2.1) e temos $v'_{j+1} > v'_j$, donde segue a afirmação.

Por outro lado $I^+_{T'}(v')$ está ordenado, pela justificativa análoga ao feito em $I^+_{T'}(x')$, pois ambos são tratados pelas operações das linhas 9 e 10. Deste último fato e afirmação anterior garantimos a ordenação de $I_{T'}(v')$, donde segue o resultado.

□

Teorema 3.3.1. *Seja $T \in \mathbb{T}_n$. O retorno do procedimento ArvoreSeguinte(T) é a árvore imediatamente maior que T em $\mathbb{T}_n$, caso seja **null** então T é o maior elemento do conjunto.*

Demonstração. Se o procedimento retorna ***null*** , o resultado é garantido pelo lema 3.3.1. Caso contrário, considere T' o retorno do algoritmo.

Como a árvore T' está ordenada pela proposição anterior, podemos efetuar a comparação entre T e T' com a disposição dos vértices retornada pelo algoritmo. Considere v o pivô de T e v' seu correspondente em T', em geral para cada x em T denotaremos por x' o seu correspondente em T', caso exista, segundo as operações nas linhas 9 e 10, mesma notação usada na prova da proposição anterior.

Afirmação: $T' > T$.

Se $v = raiz(T)$ então é claro que $T' > T$. Suponha o contrário escrevemos $I_T(v) = \{v_1 \leq \ldots \leq v_k \leq v_{k+1} \leq \ldots \leq v_h\}$ e $I_{T'}(v') = \{v'_1 \leq \ldots \leq v'_k \leq v'_{k+1} \leq \ldots \leq v'_h\}$, onde $v_k := v$ e $v'_k := v'$.

Pela linha 4 e proposição 3.3.1 temos $v' > v$. Além disso temos $v_i \backsimeq v'_i$ para $1 \leq i < k$, pois o algoritmo não faz alteração nos vértices de $I_T^-(v_k)$. Destes fatos garantimos que $pai(v') > pai(v)$, pelo caso 4.3b da definição 3.2.1.

Seja $x = pai(v)$, escreva $I_T(x) = \{x_1 \leq \ldots \leq x_j \leq \ldots \leq x_u\}$ e $I_{T'}(x') = \{x'_1 \leq \ldots \leq x'_j \leq \ldots \leq x'_u\}$, onde $x_j := x$ e $x'_j := x'$. Com raciocínio semelhante ao feito acima temos $x_i \backsimeq x'_i$ para $1 \leq i < j$, então pelo caso 4.3b da definição 3.2.1 garantimos $pai(x') > pai(x)$, uma vez que $x' > x$. Em geral o mesmo raciocínio pode ser feito de forma sucessiva para cada ancestral x de v, obtendo sempre $x' > x$. Ao alcançar o caso $x = raiz(T)$ garantimos que $T' > T$, donde segue a afirmação feita.

Vamos agora provar que T' é imediatamente maior que T em $\mathbb{T}_n$.

Inicialmente o algoritmo toma o pivô v de T e constrói v' cuja partição induzida é imediatamente maior que a de v, pela linha 5. Seja x um ancestral de v ou ainda $x = v$, então para cada $y \in I_T^+(x)$ o algoritmo constrói $y'_i \in I_{T'}(x')$ da seguinte forma: se $y \in I_T^-(x)$ então y' é equivalente a x', se $y \in I_T^+(x)$, este é um vértice esgotado pela definição de pivô, então y é construído de forma que este seja o menor vértice possível, conforme detalhado abaixo e semelhante a prova da proposição anterior.

1. Se $x \leq y$ e $l(x) = l(y)$ então $x' \backsimeq y'$ (linha 9).
2. Se $x < y$ e $l(x) < l(y)$ então y' é construído a partir da menor partição $Part(l(y))$, obtendo $x' < y'$ (linha 10).

Como $j = min\{i; x_i \not\backsimeq x'_i\}$ e $x'_j > x_j$, então $pai(x') > pai(x)$, por definição. Além disso, pelos casos vistos acima, para cada $i \in \{j+1, \ldots, u\}$ temos x_i esgotado em T e x'_i em T' construído de forma que seja o menor possível com $l(x_i)$ folhas, então $pai(x')$ é imediatamente maior que $pai(x)$, dentre todos os vértices com $l(pai(x))$ folhas induzidas.

Como esse argumento vale para cada x ancestral de v ou $x = v$, então T' é imediatamente maior que T em $\mathbb{T}_n$. □

Exemplo 3.3.7. *Abaixo ilustramos as árvores $T, T', T'' \in \mathbb{T}_{40}$, onde e T'' imediatamente maior que T' e este por sua vez imediatamente maior que T, onde T' e T'' são retornadas pelo algoritmo 3.8. Na ilustração abaixo destacamos o vértice pivô de cada árvore.*

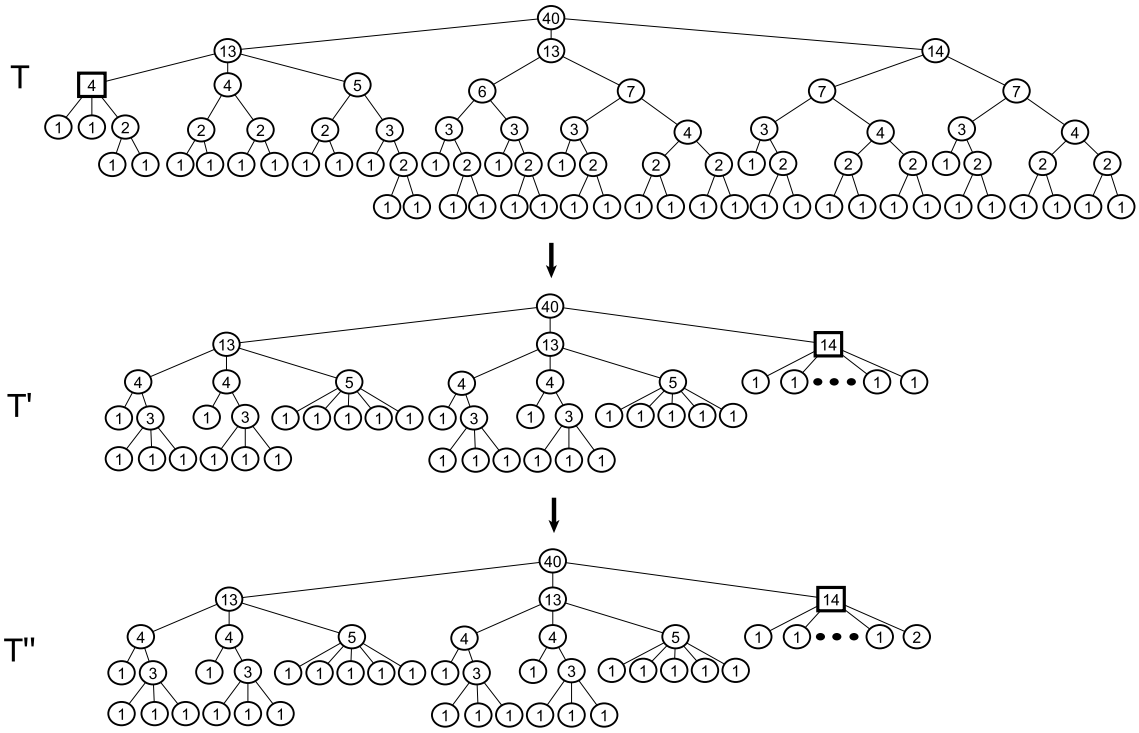

Figura 3.13: Árvores $T < T' < T''$ imediatamente maior

Proposição 3.3.5. *Se $T \in \mathbb{T}_n$ o algoritmo ArvoreSeguinte(T) tem complexidade $O(n)$.*

Demonstração. O pior caso ocorre quando a árvore possui pivô v.

O procedimento de busca do pivô executado na linha 2 tem complexidade $O(n)$ pelo fato 3.3.3. Bem como as linhas 4 e 5, que possuem complexidade $O(l(v)) = O(n)$.

Se $v = raiz(T)$ então o *loop* das linhas $7 - 13$ não será executado e o procedimento é realizado em tempo $O(n)$.

Suponha então que $v \neq raiz(T)$ e seja $P_v : (v = v_0, v_1, \ldots, v_{k-1}, v_k = raiz(T))$ o caminho do pivô até a raiz da árvore. Para cada x de P_v o *loop* interno (linha 8) executará para cada $y \in I^+(x)$ as operações da linha 9 e 10, escrevendo $I(x) = \{x_1, \ldots, x = x_j, \ldots, x_k\}$, a complexidade deste *loop* é $\sum_{i=2}^{k} O(l(x_i)) = O(l(pai(x))$, onde a igualdade decorre do fato 3.1.2.

O *loop* externo (linha 7) executa o *loop* interno para cada x de P_v (linha 12), então seguindo a ideia desenvolvida no parágrafo anterior concluímos o seguinte raciocínio:

- Para $x = v_0$ o processo da linha 8 é feito para cada $I^+(v_0)$ cujo custo é $O(l(v_1))$;
- Para $x = v_1$ o tempo de execução do *loop* externo é $O(l(v_1))$, referente à iteração anterior, somado a $\sum_{w \in I^+(v_1)} O(l(w)))$, o que totaliza tempo $O(l(v_2))$;
- Com mesmo raciocínio, para $x = v_i$ o tempo de execução é

$$O(l(v_i)) + \sum_{w \in I^+(v_i)} O(l(w))) = O(l(v_{i+1})),$$

onde a igualdade decorre do fato 3.1.2. Note que a primeira parcela é referente a complexidade das i iterações anteriores. No esquema abaixo, os tracejados indicam as subárvores que serão modificadas pelo procedimento.

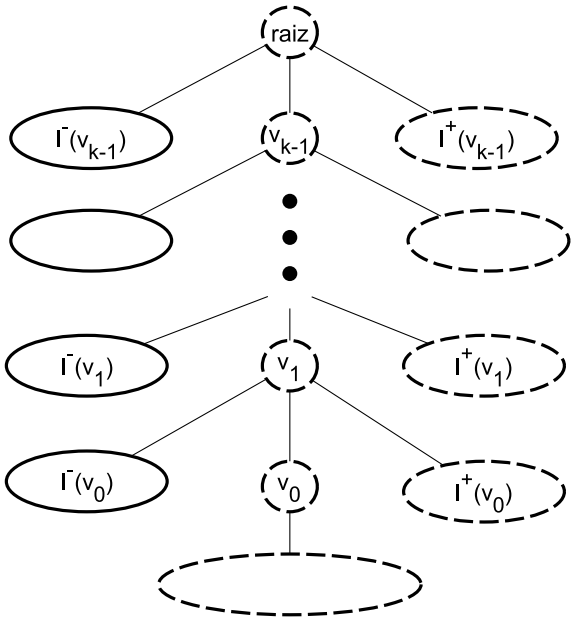

Figura 3.14: Esquema da proposição 3.3.5

O processo finaliza ao alcançar o último ancestral $x = v_k$, portanto o *loop* tem complexidade $O(l(v_k)) = O(n)$. Donde segue o resultado. □

A geração dos elementos de $\mathbb{T}_n$ será feita da seguinte forma: tomamos a menor árvore de $\mathbb{T}_n$ e a partir desta obtemos a árvore imediatamente maior por sucessivas aplicações do algoritmo 3.8, até alcançar a maior árvore do conjunto (veja o exemplo 3.3.5), indicada pela ausência de pivô.

Note que a entrada do algoritmo 3.8 exige que a árvore esteja ordenada, então a princípio seria necessário, em cada iteração, ordenar a árvore obtida para então fazer aplicar o algoritmo 3.8, o que seria extremamente custoso em termos de tempo de execução. Porém desenvolvemos o algoritmo de forma a manter a ordenação da árvore, conforme garantimos na proposição 3.3.4. Portanto é necessário a ordenação apenas na primeira árvore, que é trivial (fato 3.2.2) que e as demais terão a ordenação mantida, garantindo a eficiência do procedimento. A seguir descrevemos o algoritmo de geração em $\mathbb{T}_n$.

Algorithm 3.9 Geração de $\mathbb{T}_n$

Input: inteiro $n \geq 2$

Output: $\mathbb{T}_n = \{T_1, T_2, \ldots, T_M\}$

```
1: procedure GeracaoArvores(n)
2:     Seja T_1 a menor árvore de T_n.
3:     T_1 ← Rotular(T_1)
4:     i ← 1
5:     repeat
6:         T_{i+1} ← ArvoreSeguinte(T_i)
7:         i ← i + 1
8:     until T_{i+1} = null
9: end function
```

A corretude do algoritmo é provada a seguir.

Proposição 3.3.6. *Dado n inteiro positivo, o algoritmo acima é capaz de gerar todos os elementos $T_1, T_2, \ldots T_M$ do conjunto $\mathbb{T}_n$, sem geração de árvores isomorfas. Além disso, a geração ocorre em ordem crescente.*

Demonstração. Como o algoritmo inicia na menor árvore de $\mathbb{T}_n$ e sua condição de parada é quando alcança a maior árvore T_n, então pelo teorema 3.3.1 garantimos que o procedimento enumera todos os elementos de $\mathbb{T}_n$. Além disso as árvores são geradas em ordem crescente $T_1 < T_2 < \ldots < T_M$, onde todas são não isomorfas dois a dois, pelo lema 3.2.1. □

É claro que o algoritmo 3.9 tem complexidade total *não polinomial*, uma vez que o número de árvores de $\mathbb{T}_n$ (cografos n vértices) é exponencial. Contudo uma medida conveniente de eficiência para o algoritmo *GeracaoArvores*(n) é através do seu *atraso* (ou *delay*), isto é, o tempo entre a geração de duas árvores consecutivas.

Proposição 3.3.7. *Dado inteiro $n \geq 2$, a complexidade do tempo de atraso do algoritmo 3.9 é $O(n)$*

Demonstração. Entre as árvores T_i e T_{i+1}, executamos o algoritmo 3.8 cujo tempo de execução é $O(n)$, pela proposição 3.3.5. Em particular, entre as árvores T_1 e T_2, é executado o algoritmo $Rotular(T_1)$, que também tem custo $O(n)$. Donde segue o resultado. □

Exemplo 3.3.8. *Abaixo representamos os elementos de $\mathbb{T}_4$, retornado pelo algoritmo $GeracaoArvores(4)$.*

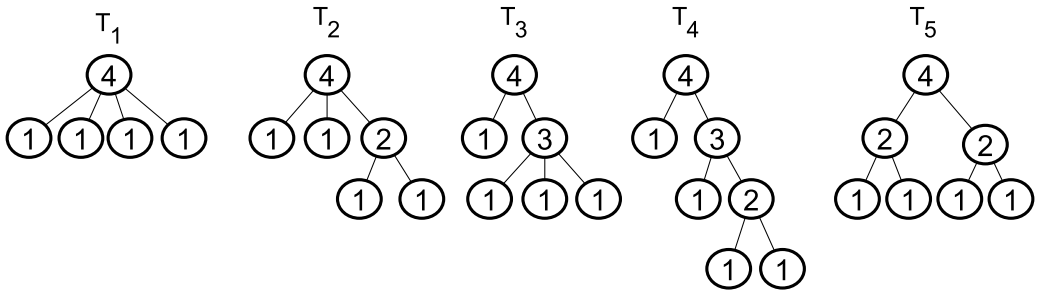

Figura 3.15: Geração de $\mathbb{T}_4$

3.4 Geração de Cografos

Agora já temos toda teoria necessária desenvolvida para escrevermos o algoritmo gerador de cografos, isto é: Dado um inteiro positivo n construir um algoritmo que gere todos os cografos com n vértices, de forma que o k-ésimo cografo gerado não seja isomorfo a nenhum dos $k-1$ cografos gerados anteriormente.

Denotaremos por $\mathscr{C}$ o conjunto de todos os cografos, tal conjunto pode ser particionado como $\mathscr{C} = \bigcup_{n=1}^{\infty} \mathscr{C}_n$, onde $\mathscr{C}_n = \{G \in \mathscr{C}; |V(G)| = n\}$. Na definição a seguir estabeleceremos uma ordenação total para os elementos de $\mathscr{C}_n$, baseado na ordenação das respectivas coárvore, segundo a definição 3.2.3.

Definição 3.4.1. *Sejam $G_1, G_2 \in \mathscr{C}_n$, e considere T_1 e T_2 suas respectivas coárvores ordenadas enraizadas em r_1 e r_2, respectivamente. Definimos:*

- *se $T_1 > T_2$ então $G_1 > G_2$*
- *se $T_1 \backsimeq T_2$, então:*
  - *se r_1 e r_2 são simultaneamente tipo-1 (ou tipo-0) então $G_1 \backsimeq G_2$*
  - *se r_1 é tipo-1 e r_2 é tipo-0 então $G_1 > G_2$*

O algoritmo gerador de cografos é similar ao algoritmo 3.9, conforme escrevemos abaixo.

Algorithm 3.10 Gerador de cografos

Input: Inteiro $n \geq 2$

Output: Geração de todos cografos com n vértices.

```
1:  procedure GeradorCografos(n)
2:      Seja T_1 a menor árvore de 𝕋_n
3:      Rotular(T_1)
4:      i ← 1
5:      repeat
6:          Seja G_i cografo associado a T, cuja raiz é tipo-0.
7:          Seja G'_i cografo associado a T, cuja raiz é tipo-1.
8:          T_{i+1} ← ArvoreSeguinte(T_i)
9:          i ← i + 1
10:     until T_i = null
11: end function
```

Teorema 3.4.1. *Dado inteiro $n \geq 2$, o algoritmo $GeradorCografos(n)$ gera todos os elementos de $\mathscr{C}_n$ de forma que todos os cografos gerados são dois a dois não isomorfos. Além disso os cografos são gerados em ordem crescente.*

Demonstração. Pela mesma justificativa aplicada na prova da proposição 3.3.6, garantimos que durante a execução do algoritmo $GeradorCografos(n)$, para $n \geq 2$, são gerados todos os elementos de $\mathbb{T}_n$ em ordem crescente e sem isomorfismos. Como cada árvore está associada a exatamente dois cografos (G e $\overline{G}$) e a proposição 3.2.1 garante que cada cografo possui uma única coárvore ordenada, então o procedimento em questão gera todos os cografos de n vértices, sem geração de cografos isomorfos.

Além disso, seja $\mathscr{C}_n = \{G_1, G'_1, G_2, G'_2, \ldots, G_M, G'_M\}$ a geração realizada pelo algoritmo.

- Para ß $\in \{1, \ldots, M\}$ temos $G_i < G'_i$, pois $T(G_i) \trianglelefteq T(G'_i)$ e $raiz(T(G_i))$ é *tipo*-0 enquanto a $raiz(T(G'_i))$ é *tipo*-1, pelas linhas 6 e 7.
- Para ß $\in \{1, \ldots, M-1\}$ temos $G'_i < G_{i+1}$, pois pois $T(G'_i) < T(G_{i+1})$ pela linha 8 e teorema 3.3.1.

Logo $G_1 < G'_1 < G_2 < G'_2 < \ldots < G_M < G'_M$. □

Exemplo 3.4.1. *Abaixo estão representados em ordem crescente todos os cografos com* 4 *vértices, gerados por* $GeradorCografos(4)$.

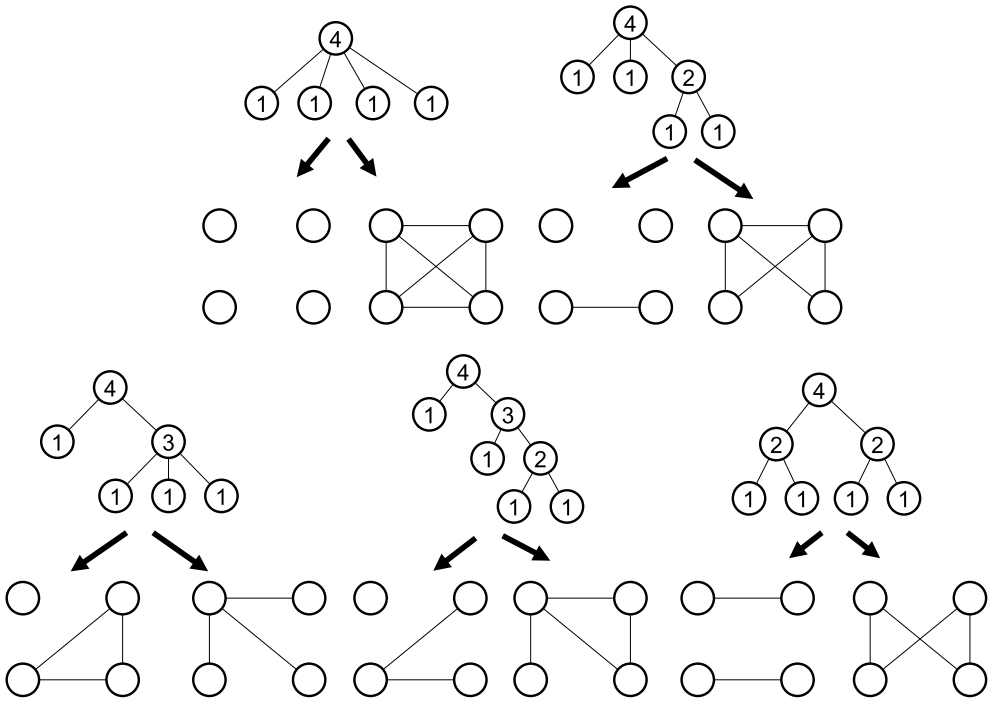

Figura 3.16: Cografos com 4 vértices

A proposição abaixo mostra a eficiência do gerador de cografos.

Proposição 3.4.1. *O algoritmo* $GeradorCografos(n)$ *possui atraso* $O(n)$.

Demonstração. Direto da definição de coárvore vemos que a operação de construção do cografo a partir da coárvore (linhas 6 e 7) é feito em tempo $O(n)$. Além disso a proposição 3.3.5 garante que a operação da linha 8 também tem custo linear. Logo a geração entre dois cografos consecutivos de $\mathscr{C}_n$ é feito em tempo $O(n)$. □

Para esclarecer melhor a importância do algoritmo acima listaremos suas características. Dado um n inteiro, o algoritmo $GeradorCografos(n)$:

- É capaz de gerar todos os cografos com n vértices;
- Possui atraso linear em n, isto é, o tempo de geração entre um cografo e o consecutivo é $O(n)$.
- Não gera cografos isomorfos em nenhum momento, o que descarta a necessidade de uma varredura em busca de cografos duplicados na geração. Este fato é garantido pelo Lema 3.2.1 e Proposição 3.3.6.

É natural nos depararmos com possíveis aplicações do algoritmo gerador de cografos, uma vez que esta classe é fortemente estudada e tal procedimento permite formulação de novas conjecturas, uma vez que podem ser testadas propriedades em todos os cografos com determinado número de vértices. Pelos mesmos motivos é também um facilitador na busca de contra-exemplos para conjecturas nesta família. Vale ressaltar que muitos parâmetros para cografos são obtidos diretamente da coárvore (como número cromático, b-cromático, número clique), o que torna a aplicação ainda mais evidente, uma vez que a construção do algoritmo é baseada unicamente na coárvore, onde a construção do cografo em si não é necessária.

Ainda observando as aplicações, sabemos que é natural restringir problemas somente aos grafos conexos. Inclusive nosso algoritmo de geração pode ser facilmente adaptado para gerar somente cografos conexos, basta retirar a linha 7 do algoritmo $GeradorCografos(n)$, conforme explicitamos abaixo.

Algorithm 3.11 Geração de cografos conexos

Input: inteiro n

Output: Geração de todos cografos conexos com n vértices.

1: **procedure** $GeracaoArvores(n)$
2: Seja T_1 a menor árvore de $\mathbb{T}_n$.
3: $T_1 \leftarrow Rotular(T_1)$
4: $i \leftarrow 1$
5: **repeat**
6: Seja G'_i cografo associado a T, cuja raiz é *tipo*-1.
7: $T_{i+1} \leftarrow ArvoreSeguinte(T_i)$
8: $i \leftarrow i + 1$
9: **until** $T_{i+1} \neq$ ***null***
10: **end function**

Motivados pela importância do algoritmo acima, fizemos a implementação na linguagem $C\sharp$ e geramos arquivos que armazenam todos os cografos conexos para um número fixado de vértices, que estão disponibilizados para download pelo link **goo.gl/9cRzPX**. O formato usado para armazenamento dos grafos é o *.graph6*, forma compacta de representar usando a escrita ASCII. Mais detalhes sobre este formato podem ser vistos em [43].

Capítulo 4

Aplicação do Gerador de Cografos em Teoria Espectral

A geração de elementos de um conjunto é útil no desenvolvimento de novas teorias, uma vez que seremos capazes de avaliar certas propriedades em todos os elementos do conjunto. Um ponto importante é a verificação de conjecturas, facilitando inclusive para construção de contra-exemplos.

Conforme descrito na introdução deste texto, na dissertação [30] é estimada uma desigualdade entre parâmetros da teoria espectral e da teoria de b-coloração, na classe dos P_4-esparsos. A saber, tal desigualdade foi o que motivou todo nosso estudo para obtenção do algoritmo de geração dos cografos, desenvolvida no capítulo 3.

Inicialmente faremos uma breve introdução com os resultados necessários para compreensão da desigualdade estimada, em seguida descreveremos os algoritmos usados na implementação, detalharemos a execução dos testes aplicados e os resultados obtidos.

4.1 Resultados preliminares

4.1.1 b-coloração

Um dos temas clássicos da teoria de grafos é o problema de coloração de vértices e suas variações. Formalmente uma ***k*-coloração** de um grafo $G(V, E)$ nada mais é que uma função sobrejetiva $c : V(G) \to \{1, \ldots, k\}$ e a **classe da cor *i*** é o conjunto $C_i = \{v \in V(G); c(v) = i\}$, onde os valores $1, \ldots, k$ são chamados cores. A restrição mais natural a ser imposta é usar k cores de forma que vértices adjacentes tenham cores distintas, chamado de ***k*-coloração própria**, o que é resolvido facilmente atribuindo uma cor para

cada vértice. Mas o problema de fato é obter a solução ótima, ou seja, o menor número de cores com essa propriedade, chamado **número cromático** do grafo, denotado por $\chi(G)$. Na dissertação [30], é tratada uma variação do problema: chamada **b-coloração**. Introduzida por Irving e Manlove [27] a b-coloração tem origem no processo chamado método *guloso* de coloração, aplicado em grafos a fim de reduzir o número de cores usadas numa coloração própria.

Dado um grafo G colorido com k cores, dizemos que um vértice é **b-dominante** se é adjacente a pelo menos um vértice de cada uma das outras cores (diferente da sua). Uma **b-coloração** é uma coloração onde cada classe de cor possui algum vértice dominante com aquela cor. Formalmente:

Definição 4.1.1. *Uma **b-coloração** é uma k-coloração tal que $\forall i \in \{1,\ldots,k\}$, $\exists u \in C_i$ (b-dominante) tal que $\forall j \in \{1,\ldots,k\}\backslash\{i\}$, $\exists v \in C_j$ com $(u,v) \in E(G)$. O número **b-cromático**, denotado por $\chi_b(G)$, é o maior k onde o grafo admite uma b-coloração com k cores.*

Exemplo 4.1.1. *Na figura 4.1 temos o grafo G à esquerda com uma 3-coloração e à direita G b-colorido com 4 cores (b-dominantes em destaque). Como não é possível diminuir o 3 na coloração própria e nem aumentar o 4 na b-coloração, então $\chi(G) = 3$ e $\chi_b(G) = 4$.*

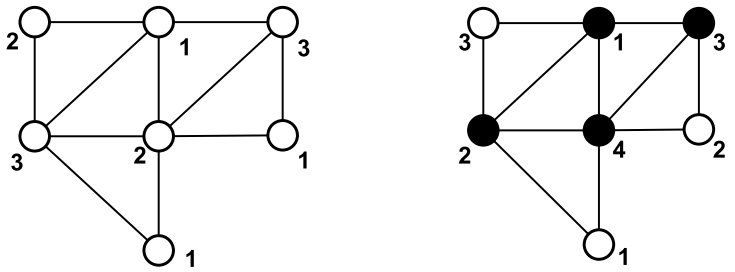

Figura 4.1: Exemplo número cromático e b-cromático

Irving e Manlove introduzem também o conceito de **m-grau** de um grafo G, denotado por $m(G)$ como o maior inteiro m tal que o grafo possua m vértices de grau maior ou igual a $m-1$. Este é um parâmetro simples de ser calculado: basta ordenar os graus dos vértices do grafo em $d(v_1) \geq \ldots \geq d(v_n)$ e obtemos $m(G) = max\{i; d(v_i) \geq i-1,\ 1 \leq i \leq n\}$. No exemplo anterior temos $m(G) = 4$.

Proposição 4.1.1. *[27] Para todo grafo G vale $\chi(G) \leq \chi_b(G) \leq m(G)$.*

Demonstração. De fato se $\chi_b(G) = k$, então então existem pelo menos k vértices b-dominantes em G, onde cada um tem grau pelo menos $k-1$, donde segue a segunda desigualdade. Para a primeira basta observar que $\chi(G)$ é o menor número de cores tal que grafo admite coloração própria. □

4.1.2 Teoria Espectral dos Grafos

O assunto principal da dissertação citada é na Teoria Espectral de Grafos, que consiste em estudar as propriedades estruturais de grafos através de suas representações matriciais e seus respectivos autovalores, que tem como marco inicial a tese de doutorado de Cvetković, 1971. A representação matricial mais comum de um grafo é por meio da matriz de **adjacência**.

Antes dos conceitos iniciais da teoria espectral de grafos, vale lembrar o conceito de autovetores e autovalores em Álgebra Linear.

Dada uma matriz A real quadrada de ordem n, diremos que o vetor não-nulo $v \in \mathbb{R}^n$ é um autovetor de A se existe λ real tal que $A.v = \lambda.v$, neste caso diremos que λ é autovalor de A associado ao autovetor v.

Então obter autovalores e autovetores de uma matriz consiste em resolver o sistema homogêneo:

$$(A - \lambda.\mathbb{I}_n).v = \mathbf{0}, \tag{4.1}$$

onde $\mathbb{I}_n$ é a matriz identidade de ordem n.

A expressão $p(\lambda) = det(A - \lambda.\mathbb{I}_n)$ é um polinômio de grau n em λ, cujas raízes são os autovalores da matriz A.

E podemos introduzir a definição da matriz de adjacência.

Definição 4.1.2. *Dado um grafo G com $V(G) = \{v_1, \ldots, v_n\}$ definimos a sua **matriz de adjacência** como a matriz $A(G)$ cuja entrada a_{ij} vale 1 se $(v_i, v_j) \in E(G)$ e vale 0, caso contrário.*

*Definimos os autovalores do grafo G como os autovalores da matriz $A(G)$. O **espectro** do grafo é o conjunto formado pelos seus autovalores e suas respectivas multiplicidades. O maior autovalor é chamado índice e denotado por $\lambda(G)$.*

Exemplo 4.1.2. *Grafo H e matriz $A(H)$:*

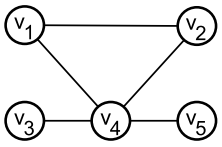

Figura 4.2: grafo exemplo

$$A(H) = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

O espectro de H é $\{2,342... \quad ; \quad 0,470...; \quad 0 \quad ; \quad -1 \quad ; \quad -1,813...\}$.

Observe que a matriz de adjacência é sempre real e simétrica então todos seus autovalores são reais, mais detalhes podem ser vistos em [26].

Vale ressaltar que a obtenção do espectro de um grafo G, com n vértices, não é uma tarefa trivial, uma vez que envolve o cálculo de um determinante de ordem n seguida pela obtenção de raízes de um polinômio de grau n.

Abaixo enunciamos dois resultados conhecidos da literatura que valem ressaltar, mais detalhes acerca da teoria espectral de grafos pode ser encontrado em [9]:

Proposição 4.1.2. *Os autovalores são números irracionais ou inteiros.*

Demonstração. Pela observação anterior basta provar que não podem ser números racionais não inteiros. Suponha que a fração $\frac{c}{d} \in \mathbb{Q}$ é autovalor do grafo, onde podemos tomar c e d primos entre si. Por outro lado como todas as entradas da matriz são 0 e 1, então seu polinômio característico é da forma

$$p(x) = x^n + a_{n-1}x^{n-1} \ldots + a_1 x + a_0 \in \mathbb{Z}[x]$$

.

Mas $\frac{c}{d}$ é raiz do polinômio, então

$$(\frac{c}{d})^n + a_{n-1}(\frac{c}{d})^{n-1} \ldots + a_1\frac{c}{d} + a_0 = 0, \text{ donde}$$

$$c^n = -(a_{n-1}c^{n-1}.d + a_{n-2}c^{n-2}.d^2 + \ldots + a_1.c.d^{n-1} + a_0.d^n).$$

Como d divide o segundo membro da igualdade então $d|c^n$, mas c e d são primos entre si, então a única opção é $d = 1$, ou seja, $\frac{c}{d} \in \mathbb{Z}$. □

Proposição 4.1.3. *O espectro de um grafo é bem definido.*

Demonstração. Dado um grafo G, podemos correspondê-lo à matrizes de adjacência distintas, uma vez que a definição depende da rotulação atribuída aos vértices. Seja A matriz de adjacência do grafo G e após uma permutação de vértices obtemos a matriz de adjacência B do mesmo grafo, tal matriz nada mais é que permutações de linhas e colunas da matriz A, isto é, existe uma matriz de permutação P tal que $B = P.A.P^{-1}$, então

$$p_B(x) = |B - xI| = |P.A.P^{-1} - xI| = |P.(A - xI).P^{-1}| = |A - xI| = p_A(x).$$

Logo as matrizes B e A têm o mesmo espectro. Em contrapartida é fácil ver que dada uma matriz de adjacência esta é associada a um único grafo.

□

4.2 A desigualdade proposta

O estudo de cotas envolvendo o número cromático de um grafo e o seu índice desperta interesse há mais de 40 anos, como pode ser visto em [13]. Em particular Wilf provou o seguinte teorema:

Teorema 4.2.1. *[57] Todo grafo G satisfaz $\chi(G) \leq \lambda(G) + 1$.*

Em relação ao número b-cromático também já existem resultados envolvendo cotas, como em [1], mas ainda não há estudos sobre cotas envolvendo os autovalores de um grafo. Na dissertação [30] foram provados alguns resultados que estabelecem cotas para o número b-cromático em função do índice. Dentre os resultados ressaltamos o seguinte:

Proposição 4.2.1. *[30] Todo grafo aranha G cuja cabeça induz um grafo r-regular satisfaz $\chi_b(G) \leq \lambda(G) + 1$.*

Motivado pela estreita relação entre os grafos aranha e os P_4-esparsos, estabelecida em [29], foi levantada a seguinte questão na dissertação: *Todo grafo P_4-esparso conexo G satisfaz $\chi_b(G) \leq \lambda(G) + 1$.*

Vale lembrar que um grafo é P_4-esparso se a cada cinco vértices existe no máximo um P_4 induzido. Naturalmente esta classe contém a dos cografos. A questão acima sugere

uma desigualdade razoável no sentido que a igualdade é válida para grafos completos, pois são P_4-esparsos e satisfazem $\lambda(K_n) = n - 1$ e $\chi_b(K_n) = n$.

Neste trabalho focaremos no caso restrito dos cografos, onde faremos uso do algoritmo 3.10 que desenvolvemos na seção 3.4. Isto é, verificaremos se a seguinte afirmação é verdadeira:

$$\textit{Todo cografo conexo } G \textit{ satisfaz } \chi_b(G) \leq \lambda(G) + 1. \quad \textbf{(}\boldsymbol{\Delta}\textbf{1)}$$

Um grafo é dito **b-perfeito** se qualquer subgrafo induzido H de G satisfaz $\chi(H) = \chi_b(H)$ (inclusive o próprio G). Então, pelo teorema de Wilf descrito acima, a afirmação é facilmente satisfeita para grafos b-perfeitos, restando verificar nos grafos não b-perfeitos. No caso dos cografos nos apoiamos no seguinte resultado:

Teorema 4.2.2. *[24] Um cografo é b-perfeito se, e somente se, for livre de* $3P_3$ *e* $2D$.

O grafo D mencionado no teorema anterior é chamado grafo diamante e é obtido pelo K_4 após retirada de uma aresta. Abaixo ilustramos os grafos $3P_3$ e $2D$

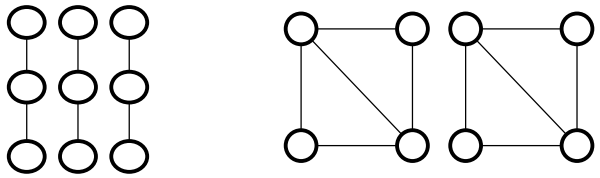

Figura 4.3: Grafos $3P_3$ e $2D$

4.3 Algoritmos implementados

A fim de verificar a existência de contra-exemplos para a afirmação **(Δ1)**, efetuamos testes computacionais fazendo uma busca nos cografos conexos. Nas subseções seguintes detalharemos a estrutura de dados utilizada, bem como descrevemos os algoritmos implementados.

4.3.1 Estrutura de dados do cografo

Como todo cografo é representado por uma árvore e os parâmetros que desejamos podem ser obtidos diretamente da árvore, é natural representarmos o cografo através de sua

árvore, que por sua vez será implementada através de listas encadeadas de vértices, cuja estrutura básica descrevemos abaixo.

Algorithm 4.12 Estrutura do Vértice

```
1:  procedure Vertice
2:      Vertice Pai;
3:      Vetor < Vertice > Filhos;
4:      double Diag;
5:      integer FolhasInduzidas;                    ▷ Valor l(v)
6:      integer label;                  ▷ Nó interno: 0, 1. Folha: 2
7:      integer Cromatico;
8:      integer Nivel;
9:      integer Grau;
10: end function
```

Dada uma árvore T e um vértice estruturado como acima, detalhamos abaixo a função de cada campo.

- vértice *Pai* armazenará $pai(v)$;
- Vetor *Filhos* armazenará $Filhos(v)$;
- escalar *Diag* será visto posteriormente, cuja utilidade é obtenção de estimativa para o índice de T;
- inteiro *FolhasInduzidas* armazena o valor $l(v)$;
- inteiro *label* armazena se o vértice é interno (*tipo*-1 ou *tipo*-0) ou uma folha (denotado por $tipo - 2$)
- inteiro *Cromatico* armazena o número cromático do cografo cuja coárvore é $T(v)$, será visto em detalhes na seção seguinte.
- inteiro *Nivel* armazena o valor $nivel(v)$;
- inteiro *Grau* armazena o grau do vértice v no cografo referente a T. Claro que este campo só faz sentido nos vértices folha.

A árvore é acessada através de seu vértice raiz, que permite operar e obter informações de toda a árvore.

O acesso a qualquer campo da estrutura vértice, será denotado por ".". Por exemplo ao acessar o *label* de um vértice v, usaremos a notação $v.label$.

O preenchimento do campo *FolhasInduzidas* foi feito de forma recursiva através do procedimento descrito abaixo:

Algorithm 4.13 Calcula o número de folhas induzidas numa árvore

Input: Vértice v de uma árvore T

Output: Número de folhas induzidas pela subárvore $T(v)$

1: **procedure** $NumDeFolhas(v)$
2: | **if** v não é folha **then**
3: | | $v.FolhasInduzidas = \sum_{x \in filhos(v)} NumDeFolhas(x)$
4: | **else**
5: | | $v.FolhasInduzidas = 1$
6: | **end if**
7: | **return** $v.FolhasInduzidas$
8: **end function**

O preenchimento do campo *nivel* é a aplicação direta da definição, ou seja, a raiz tem nível 1 e a cada novo vértice v instanciado, atribuímos ao campo $v.nivel \leftarrow v.pai.nivel+1$.

A seguir detalhamos o preenchimento do campo *Grau*. Observe que não nos referimos ao grau do vértice na árvore, mas sim no vértice correspondente no cografo associado a esta árvore, portanto este campo só faz sentido nas folhas da árvore, para os demais vértices o campo é útil no procedimento. Vale ressaltar que estamos calculando diretamente na coárvore um parâmetro do cografo, o que fornece eficiência no processo. A construção do algoritmo é baseada no fato 2.3.1.

Algorithm 4.14 Calcula o grau (cografo) de cada vértice

Input: Vértice v de uma coárvore T

Output: Calcula o grau de cada vértice na subárvore $T(v)$

```
1:  procedure CalculaGrau(v)
2:      if v.label = 1 then
3:          for all w ∈ v.Filhos do
4:              aux ← Σ_{x∈I(w)\{w}} x.FolhasInduzidas
5:              w.Grau ← v.Grau + aux
6:              CalculaGrau(w)
7:          end for
8:      else
9:          if v.label = 0 then
10:             for all w ∈ v.Filhos do
11:                 w.Grau ← v.Grau
12:                 CalculaGrau(w)
13:             end for
14:         end if
15:     end if
16: end function
```

Omitiremos mais detalhes sobre o preenchimento dos demais campos, pois fogem do objetivo do presente texto. Daqui em diante assumiremos que todo vértice já possui seus campos *FolhasInduzidas*, *Nivel*, *Pai*, *Filhos*, *label* e *Grau* devidamente preenchidos.

4.3.2 Cálculo do b-cromático

O cálculo do número b-cromático é um problema NP-difícil para classes gerais de grafos, como provado em [27]. Contudo Bonomo *et al* provam em [6] que este cálculo é polinomial para cografos, usaremos a estratégia proposta no artigo para obter o número b-cromático de forma eficiente. Antes de descrevermos o procedimento, é necessário introduzir o conceito de vetor b-dominante.

Definição 4.3.1. *[6] Dado um grafo G, o vetor b-dominante dom_G é um vetor, onde para cada t, $\chi(G) \leq t \leq |V(G)|$, a entrada $dom_G[t]$ é o maior número de classes de cores que possuem vértice b-dominante dentre qualquer coloração própria de G com t cores. Para valores de t fora da intervalo, observe que $dom_G[t] = 0$ para $t > |V(G)|$ e $dom_G[t]$ não faz*

sentido para valores $t < \chi(G)$.

Exemplo 4.3.1. *O vetor b-dominante do grafo abaixo é* $(3, 3, 2, 0)$, *onde* $\chi(G) = 3$.

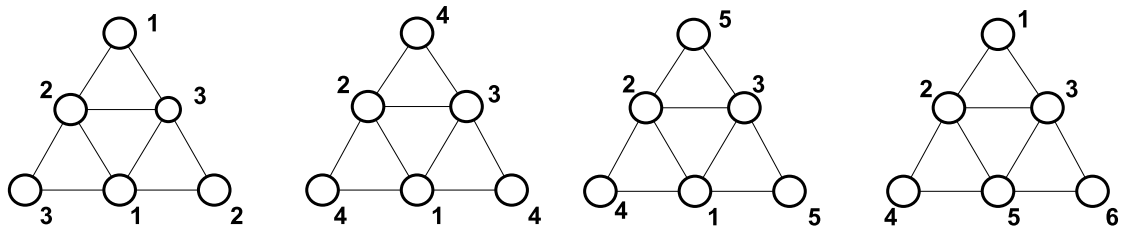

Figura 4.4: Grafo colorido com 3,4,5 e 6 cores, nesta ordem.

Então observe que o número b-cromático de um grafo G é o maior inteiro k onde $dom_G[k] = k$. Além disso Bonomo *et al.* provam o seguinte lema:

Lema 4.3.1. *[6] Se* G *é um grafo e* $t \geq \chi(G)$, *então ou* $dom_G[t{+}1] = t{+}1$ *ou* $dom_G[t{+}1] \leq dom_G[t]$.

Com o lema acima e a observação anterior, fica natural um procedimento para o cálculo do b-cromático de um grafo através de seu vetor b-dominante: basta tomar o inteiro $t \geq \chi(G)$ tal que $dom_G[t+1] \neq (t+1)$. Então nos deparamos com um novo problema: como calcular o vetor b-dominante de cografos?

Uma boa estratégia para calcular um parâmetro em um cografo é estudar o comportamento deste nas operações de união e *join* (conforme algoritmo 4.14) e em seguida fazer o cálculo de forma recursiva a partir de sua coárvore. Isto foi exatamente o que Bonomo *et al.* propõem para o cálculo do vetor b-dominante, a seguir:

Teorema 4.3.1. *[6] Sejam* $G_1(V_1, E_1)$ *e* $G_2(V_2, E_2)$ *dois grafos tais que* $V_1 \cap V_2 = \emptyset$. *Se* $G = G_1 \cup G_2$ *e* $t \geq \chi(G)$, *então*

$$dom_G[t] = min\{t, dom_{G_1}[t] + dom_{G_2}[t]\}$$

Teorema 4.3.2. *[6] Sejam* $G_1(V_1, E_1)$ *e* $G_2(V_2, E_2)$ *dois grafos tais que* $V_1 \cap V_2 = \emptyset$. *Se* $G = G_1 \vee G_2$ *e* $\chi(G) \leq t \leq |V(G)|$. *Seja ainda* $a = max\{\chi(G_1), t - |V_2|\}$ *e* $b = \{|V_1|, t - \chi(G_2)\}$. *Então* $a \leq b$ *e*

$$dom_G[t] = max\{dom_{G_1}[j] + dom_{G_2}[t-j]; a \leq j \leq b\}$$

É fácil verificar que para quaisquer grafos G_1 e G_2 o número b-cromático do *join* destes é calculado de forma direta, a saber $\chi_b(G_1 \vee G_2) = \chi_b(G_1) + \chi_b(G_2)$. Mas se nos basearmos somente nesta equação não conseguiremos calcular o b-cromático do cografo,

pois a operação de união não depende exclusivamente do b-cromático dos grafos que o compõem, mas sim de todo o vetor b-dominante do grafo. Observe o seguinte exemplo:

Exemplo 4.3.2. *O grafo* ***estrela*** *é definido por* $S_n = K_1 \vee \overline{K_n}$. *É fácil observar que* $\chi_b(S_n) = 2$, *em particular* $\chi_b(S_4) = 2$ *e* $\chi_b(2S_4) = 2$. *Porém* $\chi_b(5S_4) = 5$, *conforme pode ser visto na ilustração abaixo:*

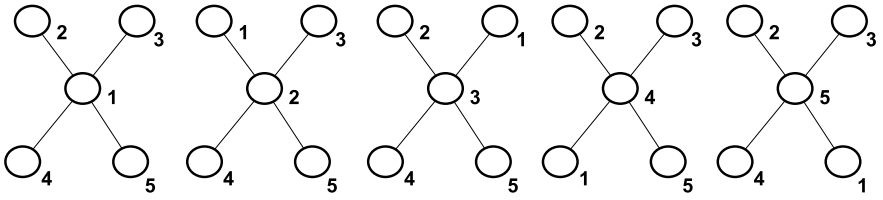

Figura 4.5: b-coloração do grafo $5S_4$

Portanto o processo de calcular o número b-cromático de uma união de grafos envolve o cálculo do vetor b-dominante, conforme detalhado no teorema 4.3.1. Isto justificativa o cálculo deste vetor quando desejamos obter o número b-cromático de um cografo. Observando o enunciado dos teoremas 4.3.1 e 4.3.2 vemos que é necessário calcular inicialmente o número cromático de cografos, onde nos apoiaremos no teorema de Corneil *et al*:

Teorema 4.3.3. *[11] Se* G *é o grafo trivial, então* $\chi(G) = 1$. *Sejam* $G_1(V_1, E_1)$ *e* $G_2(V_2, E_2)$ *dois grafos tais que* $V_1 \cap V_2 = \emptyset$, *então* $\chi(G_1 \cup G_2) = max\{\chi(G_1), \chi(G_2)\}$ *e* $\chi(G_1 \vee G_2) = \chi(G_1) + \chi(G_2)$

Então os teoremas anteriores serão nossas ferramentas para o cálculo dos números cromático e b-cromático de um cografo. Contudo, observe que os resultados são descritos para operação de exatamente dois grafos, e na coárvore é comum encontrarmos operações dentre três ou mais grafos, lembrando que as operações são representadas por nós internos da árvore. Não é difícil escrever as generalizações dos teoremas 4.3.1 e 4.3.3 para uma operação de k grafos, mas essa facilidade não ocorre com o teorema 4.3.2. Então buscamos uma solução alternativa: transformar a coárvore em uma árvore binária estrita [1] e em seguida construir procedimentos que se baseiam somente em operações dentre dois grafos.

A transformação de uma coárvore em uma árvore binária estrita deve ser feita de forma que as operações sejam mantidas. Para tal, basta observar que dados os grafos $G_1, \ldots, G_k$ vale:

$$G_1 * G_2 * \ldots * G_k = (((G_1 * G_2) * G_3) \ldots * G_{k-1}) * G_k,$$

[1] todo vértice interno possui exatamente 2 filhos

onde $*$ denota a operação *join* ou união.

A seguir escrevemos o algoritmo que converte uma árvore enraizada em v numa árvore binária estrita.

Algorithm 4.15 Converte coárvore numa binária estrita

Input: Coárvore T com raiz v

Output: Coárvore T escrita na forma binária estrita

```
1:  procedure ConverteEmBinaria(v)
2:      while |filhos(v)| ≥ 3 do
3:          Seja v' vértice.
4:          Tome a, b filhos de v.
5:          v'.pai ← v
6:          Defina tipo de v' igual ao tipo de v
7:          a.pai ← v'
8:          b.pai ← v'
9:      end while
10:     para cada u ∈ filhos(v) faça ConverteEmBinaria(u)
11: end function
```

Para uma melhor compreensão do algoritmo, segue um esquema de sua execução:

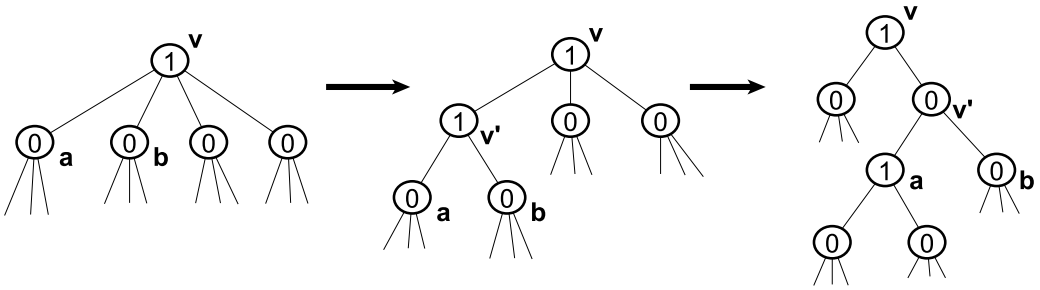

Figura 4.6: Execução do Algoritmo 4.15 sobre o vértice v

Na definição formal de coárvore, vemos que a configuração minimiza o número de nós internos, no sentido de que qualquer caminho tomado os vértices internos alternam dentre *join* e união. O que não ocorre na sua forma binária estrita, o que pode parecer estranho, pois afinal a coárvore é única a menos de permutação de vértices. O que fazemos aqui é um abuso de linguagem, pois de fato na forma binária é fácil ver que a árvore continua representando o mesmo cografo, mas formalmente não é mais uma coárvore.

A corretude do algoritmo anterior é válida, uma vez que este efetua sobre cada vértice uma reorganização dos seus filhos, através da inserção de auxiliares que não interferem na estrutura do cografo.

Fazendo uso do último algoritmo e do teorema 4.3.3, podemos escrever o processo que calcula o número cromático de um cografo através de sua coárvore. Dado um cografo G com coárvore associada T, a ideia geral do procedimento é armazenar, de forma recursiva, no campo *Cromatico* de cada vértice v o número cromático do cografo referente a coárvore $T(v)$. Logo o número cromático do cografo G é o valor armazenado na raiz de T.

Algorithm 4.16 Cálculo do número cromático

Input: Vértice v

Output: Número cromático de v

```
1:  procedure Cromatico(v)
2:      if v é folha then
3:          return 1;
4:      else
5:          if v.label = 0 then                          ▷ nó união
6:              v.Cromatico ← max{Cromatico(x); x ∈ filhos(v)}.
7:          else                                         ▷ nó join
8:              v.Cromatico ← Σ_{x∈filhos(v)} Cromatico(x).
9:          end if
10:     end if
11:     return v.Cromatico
12: end function
```

A corretude do algoritmo acima é garantida pelo teorema 4.3.3.

Com número cromático calculado e armazenado em cada vértice, podemos escrever o algoritmo recursivo que calcula a coordenada t do vetor b-dominante do vértice v, que representa o cografo referente a coárvore $T(v)$. Antes vale ressaltar que o procedimento anterior não necessita da árvore na forma binária estrita, porém no seguinte assumimos que a entrada é uma árvore com tal restrição.

Algorithm 4.17 Cálculo de coordenada do vetor b-dominante

Input: Vértice v e parâmetro t

Output: Valor $dom_G[t]$, onde G é cografo associado a $T(v)$

```
1: se (t > v.FolhasInduzidas) então return 0;
2: procedure dom(v, t)
3:     if v é folha then
4:         return 1;
5:     else
6:         Sejam x, y os filhos de v                    ▷ árvore binária
7:         if v é tipo-0 then
8:             return min{t, dom(x, t) + dom(y, t)}
9:         else
10:            a ← max{x.Cromatico, t − y.FolhasInduzidas}
11:            b ← min{x.FolhasInduzidas, t − y.Cromatico}
12:            return max{dom(x, j) + dom(y, t − j); a ≤ j ≤ b}
13:        end if
14:    end if
15: end function
```

A sua corretude é garantida pelos teoremas 4.3.1 e 4.3.2.

Com os três algoritmos acima podemos, por fim, escrever o algoritmo que calcula o número b-cromático de um grafo.

Algorithm 4.18 Cálculo do número b-cromático

Input: Coárvore T

Output: Número b-cromático do cografo G associado à T

```
1: procedure Bcromatico(T)
2:     r ← raiz(T)
3:     ConverteEmBinaria(r)
4:     se r é folha então return 1;
5:     t ← Cromatico(r)
6:     enquanto t = dom(r, t) faça t ← t + 1
7:     return t − 1;
8: end function
```

A corretude do algoritmo anterior é assegurado pelo lema 4.3.1.

4.3.3 Cálculo do índice

A obtenção do índice de um grafo G com n vértices, envolve o cálculo de determinantes de ordem n e de raízes de polinômios de grau n, o que pode ser custoso computacionalmente. Como nosso propósito é executar vários testes a fim de verificar a afirmação **(Δ1)**, o cálculo pela definição não é um caminho viável. Como só queremos verificar uma desigualdade utilizamos um método mais eficiente capaz de majorar os autovalores.

Em [28] é proposto um algoritmo de localização de autovalores em cografos. A entrada do algoritmo é uma coárvore T na forma *standard*, isto é, coárvore estruturada de forma que todas as folhas estejam no último nível. O procedimento a seguir descreve as operações que transformam uma coárvore na sua versão *standard*, cuja ideia principal nada mais é que inserir vértices intermediários que não alteram a estrutura do cografo, até que todas as folhas sejam postas no último nível.

Algorithm 4.19 Transforma em coárvore *standard*

Input: Coárvore T

Output: Coárvore T' *standard*

```
1:  procedure Standard(T)
2:      for i ← 2 to nivel(T) do
3:          for all folha v de nivel(T, i) do
4:              while v.Nivel < nitel(T) do
5:                  Seja w ← v.pai
6:                  Seja o vértice aux
7:                  Estabelece aux como filho de w
8:                  Atribua em aux o tipo oposto ao de w
9:                  Remove parentesco entre v e w
10:                 Estabelece v como filho de aux
11:                 v.Nivel ← v.Nivel + 1
12:             end while
13:         end for
14:     end for
15: end function
```

Exemplo 4.3.3. *A seguir segue um exemplo da aplicação do algoritmo acima numa coárvore*

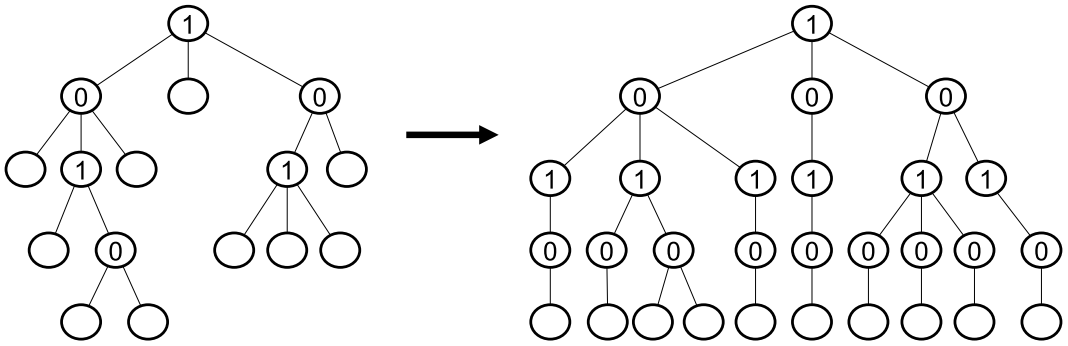

Figura 4.7: Coárvore T e sua versão *standard*

A ideia do Algoritmo proposto em [28] é a seguinte: dada uma coárvore *standard* T e um valor real x, a cada folha v de T é associado um número $diag(v)$, chamado valor diagonal, inicializado com x. Então são realizadas dois tipos de operações: atualização dos valores $diag(v)$ e remoção de adjacências em T, onde implicitamente tais processos são equivalentes a operações na matriz $A(G) + x.\mathbb{I}$. Após a execução, cada vértice terá seu valor diagonal definitivo, o que equivale a matriz estar na sua forma diagonalizada. Então é garantido que:

- A quantidade de autovalores menores que $-x$ coincide com a quantidade de folhas com valor diagonal negativo.
- A quantidade de autovalores iguais a $-x$ coincide com a quantidade de folhas com valor diagonal nulo.
- A quantidade de autovalores maiores que $-x$ coincide com a quantidade de folhas com valor diagonal positivo.

Como o nosso objetivo é verificar a validade da afirmação **(Δ1)**, ou seja, constatar se o índice de G é maior que o parâmetro $\chi_b(G)-1$, então basta aplicar o valor $x = 1-\chi_b(G)$ como entrada do procedimento e verificarmos se no final do processo existe algum vértice com valor diagonal positivo ou nulo. Caso afirmativo então existe ao menos um autovalor maior ou igual a $-x$, portanto $\lambda(G) \geq -x$ confirmando a afirmação. Por outro lado, um contra exemplo para a afirmação é um cografo G que após aplicação do procedimento com entrada x, todos os vértices possuem valor diagonal negativo, constatando que todos os autovalores, inclusive o índice, são menores que $-x$. Novamente vale ressaltar que para aplicar o algoritmo não é necessário em momento algum da estrutura do cografo, todas as operações são feitas sobre a árvore.

A seguir escrevemos o procedimento proposto no artigo. Antes observe que as n folhas da árvore estão localizadas no nível $m := nivel(T)$, as quais rotulamos em ordem decrescente de índice: $v_k, v_{k-1}, \ldots, v_2, v_1$. O valor $Diag(v)$ será armazenado no campo $v.Diag$.

Algorithm 4.20 [28] Algoritmo diagonalização

Input: Coárvore *standard* T e real x

Output: Cálculo do valor diagonal de cada vértice

procedure $LocalizacaoAutovalores(T, x)$
  For each v folha de T **do** $v_k.Diag \leftarrow x$
  for i $\leftarrow m-1$ **to** 1 **do** ▷ análise do nível i.
    while $\exists v_k,\ v_l\ (k<l)$ onde $nivel(lca(v_k, v_l)) = i$ **do**
      $\beta \leftarrow v_k.Diag$
      $\alpha \leftarrow v_l.Diag$
      if $lca(v_k, v_l)$ é *tipo*-1 **then**
        if $\alpha + \beta \neq 2$ **then**
          $v_l.Diag \leftarrow \frac{\alpha\beta - 1}{\alpha+\beta-2}$ $v_k.Diag \leftarrow \alpha + \beta - 2$
        else
          if $\beta = 1$ **then**
            $v_l.Diag \leftarrow 1$ $v_k.Diag = 0$
          else
            $v_l.Diag \leftarrow 1$ $v_k.Diag = -(1-\beta)^2$
            remova adjacências entre v_l e $lca(v_k, v_l)$
          end if
        end if
        remova adjacências entre v_k e $lca(v_k, v_l)$
      else
        if $\alpha + \beta \neq 0$ **then**
          $v_l.Diag \leftarrow \frac{\alpha\beta}{\alpha+\beta}$ $v_k.Diag \leftarrow \alpha + \beta$
        else
          if $\beta = 0$ **then**
            $v_l.Diag \leftarrow 0$ $v_k.Diag = 0$
          else
            $v_l.Diag \leftarrow \beta$ $v_k.Diag = -\beta$
            remova adjacências entre v_l e $lca(v_k, v_l)$
          end if
        end if
        remova adjacências entre v_k e $lca(v_k, v_l)$
      end if
    end while
  end for
end function

Nosso interesse é verificar se o índice de um grafo é maior que um específico valor x dado como entrada, o que é feito pelo algoritmo abaixo.

Algorithm 4.21 Verifica se o índice é maior que x

Input: Coárvore T e escalar x

Output: Valor *booleano*

```
1: procedure IndiceMaiorQue(T, x)
2:     T ← Standard(T)
3:     LocalizacaoAutovalores(T, −x)
4:     for all v folha de T do
5:         se v.Diag ≥ 0 então return true
6:     end for
7:     return false
8: end function
```

4.4 Verificação da afirmação (Δ1)

Como já citamos em outros momentos do texto, a teoria que desenvolvemos no capítulo 3 foi motivada pela verificação da afirmação **(Δ1)**. Portanto os testes foram realizados em dois momentos: Primeiro testamos a afirmação em grafos gerados de forma aleatória, pois ainda não havíamos elaborado o algoritmo gerador de cografos. Diante dos resultados obtidos, descritos na subseção 4.4.1, vimos a necessidade de gerar cografos de forma sistemática e então desenvolvemos o algoritmo gerador de cografos 3.10. O segundo momento trata da verificação de uma reformulação da afirmação **(Δ1)** em todos os cografos com fixado número de vértices, onde obtemos resultados satisfatórios, descritos na subseção 4.4.2.

Todos os testes foram feitos através dos algoritmos descritos neste trabalho, implementados na linguagem *C#*, onde foram executados numa máquina virtual fornecida pela plataforma *Google Cloud* numa máquina equipada com CPU Intel Sandy Bridge com clock 3.3 Ghz, 12GB de memória RAM e Windows 7 64 bits.

4.4.1 Testes com cografos gerados aleatoriamente

Nesta seção descreveremos os testes realizados em cografos gerados aleatoriamente. Abaixo descrevemos o algoritmo.

Algorithm 4.22 Gerador aleatório de coárvores

Input: Inteiro max e vértice v

Output: coárvore aleatória com aproximadamente max vértices

```
1:  procedure GeraCoarvoreAleatoria(v, max)
2:      if Se v não é folha then
3:          if |V(T)| < max then
4:              Seja k número aleatório do conjunto {2, 3, 4, 5}
5:              for i ← 0 to k do
6:                  Insere em v um filho w;
7:                  Decide aleatoriamente se w é interno ou folha.
8:                  if w é folha then
9:                      GeraCoarvoreAleatoria(w, max)
10:                 end if
11:             end for
12:         end if
13:         Converte v numa folha;
14:     end if
15: end function
```

Vale ressaltar que o inteiro max dado como entrada do algoritmo acima não estabelece o número máximo de vértices da árvore, mas sim uma aproximação para o número máximo. Pois dependendo do sorteio pode ocorrer de alguns vértices a mais sejam inseridos. O teste de verificação da afirmação se baseia em todos os algoritmos descritos neste capítulo, cuja ordem de execução é descrita pelo algoritmo abaixo. Que retorna um contra-exemplo para a afirmação, se encontrado.

Algorithm 4.23 Validação da afirmação **(Δ1)**

Input: Número aproximado de vértices m

Output: Coárvore T que não satisfaz a afirmação **(Δ1)**

```
1:  procedure BuscaContraExemplo(m)
2:      repeat
3:          Seja v um vértice                                   ▷ raiz
4:          T ← GeraCoarvoreAleatoria(v, m)
5:          Insere aleatoriamente o grafo 2D ou 3P3 como induzido
6:          Seja T' cópia de T
7:          x ← Bcromatico(T') − 1
8:      until !(IndiceMaiorQue(T, x))
9:      return T
10: end function
```

A operação realizada na linha 5 torna o cografo não b-perfeito, pois como já observado anteriormente, caso o cografo seja b-perfeito, a afirmação **(Δ1)** é trivialmente satisfeita. Na linha 6 é necessária uma cópia da coárvore para aplicar no algoritmo de cálculo do b-cromático, uma vez que o procedimento chamado na linha seguinte transforma numa árvore binária, tornando-a inutilizável para o algoritmo de diagonalização.

Foram gerados e testados cografos com até 20 vértices (aproximadamente), pois caso um contra-exemplo fosse encontrado, seria possível observar com mais clareza sua estrutura. Depois de gerados mais de 400.000 cografos aleatórios, o algoritmo encontrou um contra-exemplo: o cografo que denotaremos por G_a, ilustrado na figura 4.8 e possui as seguintes características:

- Possui $2D$ como induzido;
- $\lambda(G_a) \approx 6,9618$;
- $\chi_b(G) = m(G) = 8$;
- $\chi_b(G) - (\lambda(G) + 1) < 0,04$.

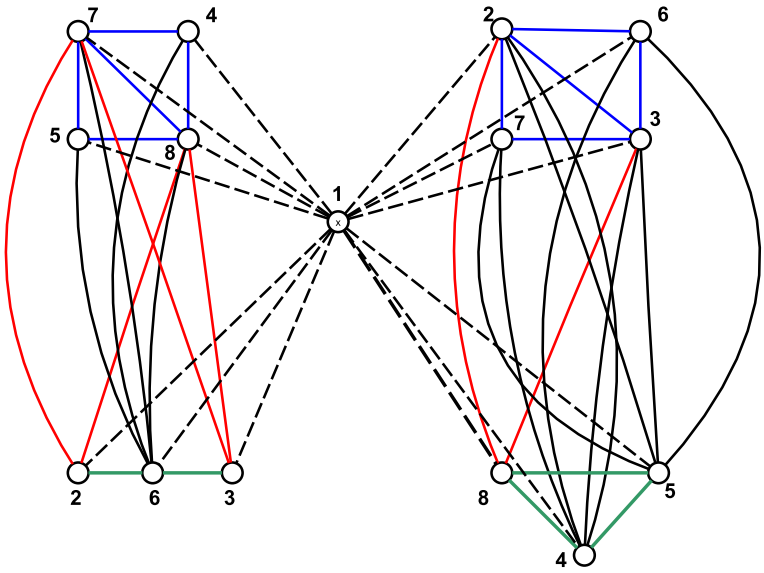

Figura 4.8: Contra-exemplo G_a

Então a afirmação **(Δ1)** proposta na dissertação [30] é falsa.

4.4.1.1 Formulação de uma nova afirmação

Com o contra-exemplo encontrado, observamos que G_a nega a desigualdade da afirmação por menos de um décimo. O que nos motiva a introdução de uma nova afirmação:

$$\textit{Todo cografo conexo } G \textit{ satisfaz } \chi_b(G) \leq \lceil \lambda(G) \rceil + 1. \qquad \mathbf{\Delta 2}$$

Onde usamos o parâmetro $\lceil \lambda(G) \rceil$ ao invés de $\lambda(G)$.

Nesta reformulação o grafo G_a não é mais um contra-exemplo.

Agora precisamos de uma forma eficiente de calcular o teto do índice de um grafo, novamente podemos usar o algoritmo de diagonalização e fazer uso do seguinte teorema para otimizar o processo.

Teorema 4.4.1. *[13] Todo grafo G satisfaz $\delta \leq \overline{d} \leq \lambda(G) \leq \Delta$, onde δ, $\overline{d}$ e Δ são os graus mínimo, médio e máximo de G, nesta ordem.*

Algorithm 4.24 Cálculo do teto do índice

Input: Coárvore T do cografo G.

Output: Valor $\lceil \lambda(G) \rceil$.

```
1: procedure TetoDoIndice(T)
2:     Seja k = ⌈d̄(G)⌉
3:     while IndiceMaiorQue(T, k) do
4:         k ← k + 1
5:     end while
6:     return k
7: end function
```

O teorema 4.4.1 é útil para otimizar o algoritmo 4.24, pois iniciamos a busca do teto do índice no valor $k = \lceil \overline{d}(G) \rceil$, não havendo necessidade de iniciarmos no $k = 0$, o que evita efetuar $\lceil \overline{d}(G) \rceil$ vezes o procedimento $IndiceMaiorQue(T, k)$. Note que no melhor caso, efetuaremos a chamada somente uma vez.

De forma semelhante ao feito no algoritmo 4.23, podemos efetuar procedimento semelhante para validar a afirmação **Δ2**. Foram testados mais de 2 milhões de cografos não b-perfeitos e nenhum contra exemplo foi encontrado, o que nos leva a acreditar que tal afirmação é verdadeira. Porém com estes testes não podemos levantar nenhuma ideia ou fato, uma vez que os cografos são gerados de forma aleatória e estamos sujeitos, inclusive, a uma geração de milhares de cografos isomorfos.

Então foi exatamente este ponto, que motivou o desenvolvimento de todo este trabalho. Inicialmente os testes eram em cografos aleatórios, sem nenhuma formação sistemática, então nos deparamos com a necessidade de elaborar uma geração dos cografos para obter resultados mais precisos. A solução para este problema foi o algoritmo 3.11: $GeracaoCografosConexos(n)$, cuja primeira aplicação ocorre na verificação da afirmação **(Δ2)**

4.4.2 Implementação do $GeracaoCografos(n)$

Nesta seção relataremos os resultados obtidos ao verificar a afirmação utilizando o algoritmo gerador de cografos 3.10.

Antes de verificar a afirmação **Δ2**, observamos que o parâmetro m-grau poderia assumir o papel do b-cromático nesta afirmação, uma vez que como visto na proposição 4.1.1, todo grafo G satisfaz a desigualdade $\chi_b(G) \leq m(G)$, então propomos a seguinte relação

para todo cografo conexo:

$$m(G) \leq \lceil \lambda(G) \rceil + 1 \tag{4.2}$$

Se a desigualdade 4.2 for mantida, obteremos trivialmente a veracidade da afirmação **Δ2**.

O parâmetro m-grau é mais simples de trabalhar, inclusive computacionalmente, uma vez que sua definição envolve apenas a sequência de graus do grafo. Em particular, no cografo podemos obter o m-grau diretamente da sua coárvore, conforme descrito no algoritmo abaixo:

Algorithm 4.25 Cálculo do m-grau de um cografo

Input: Coarvore T com n folhas

Output: m-grau do cografo referente à T

1: **procedure** $Mgrau$(T)
2: Seja seq vetor que armazena $v.Grau$ para cada folha v
3: Ordene seq decrescente
4: **return** $max\{i; seq[i] \geq i - 1; 1 \leq i \leq n\}$
5: **end function**

O processo acima é muito mais eficiente do que o cálculo do b-cromático que envolve diversas chamadas recursivas, conforme vimos na subseção 4.3.2.

Daqui em diante, a fim de não sobrecarregar a escrita, ao escrevermos sobre um parâmetro da coárvore, nos referimos ao do cografo correspondente.

Então implementamos o procedimento que utiliza o algoritmo 3.11 para verificar se a desigualdade 4.2 é mantida.

Algorithm 4.26 Verificação da desigualdade 4.2 com geração

Output: Grafo contraexemplo

1: **procedure** $ValidacaoDesigualdadeMgrau$
2: $n = 2$
3: **repeat**
4: Seja T a menor coárvore com n vértices
5: **se** $Mgrau(T) > TetoDoIndice(T) + 1$ **então stop**;
6: $T \leftarrow ArvoreSeguinte(T)$
7: **until** $T =$ ***null***
8: $n \leftarrow n + 1$
9: **end function**

E alguns contra-exemplos para a desigualdade 4.2 foram encontrados. O primeiro de todos, podemos chamar de "o menor", foi o grafo $G_b := \overline{K_7} \vee (K_6 \cup \overline{K_6})$, que possui 19 vértices, cujas características e esboço seguem abaixo:

- $\lceil \lambda(G_b) \rceil = 11$
- $m(G) = 13$
- $\chi(G) = \chi_b(G) = 7$

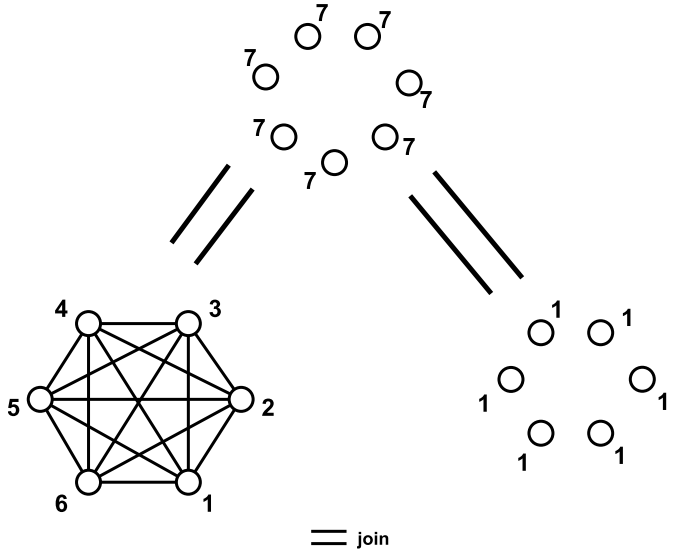

Figura 4.9: Contra-exemplo G_b

Como podemos gerar os cografos com fixado número de vértices, podemos naturalmente efetuar a contagem de quantos cografos existem. E além disso observar a quantidade de cografos que não satisfazem a desigualdade 4.2. Após realizados os testes obtemos resultados surpreendentes: constatamos que existe um número insignificante de cografos que não satisfazem a desigualdade proposta, comparado à quantidade de cografos existentes, conforme pode ser visto na tabela abaixo:

Vértices	Qtd. cografos conexos	Não vale desigualdade 4.2	proporção	Tempo
2	1	0	0 %	< 0,1s
3	2	0	0 %	< 0,1s
4	5	0	0 %	< 0,1s
5	12	0	0 %	< 0,1s
6	33	0	0 %	< 0,1s
7	90	0	0 %	< 0,1s
8	261	0	0 %	< 0,1s
9	766	0	0 %	< 0,1s
10	2312	0	0 %	< 0,1s
11	7068	0	0 %	< 0,1s
12	21965	0	0 %	< 1min
13	68.954	0	0 %	< 1min
14	218751	0	0 %	< 1min
15	699534	0	0 %	13 min
16	2253676	0	0 %	50 min
17	7305788	0	0 %	3h
18	23816743	0	0 %	9h
19	78023602	3	$4,146 \times 10^{-8}$ %	30h
20	256738751	21	$8,919 \times 10^{-8}$ %	3 dias
21	848152864	88	$1,127 \times 10^{-7}$ %	9 dias

Tabela 4.1: Quantidade de Cografos Conexos

Em contra-partida, executamos a validação da afirmação **Δ2** e obtivemos ótimos resultados. O algoritmo implementado é semelhante ao 4.26, onde buscamos otimizar os testes conforme pode ser visto no seguinte algoritmo:

Algorithm 4.27 Verificação da afirmação **Δ2**

Output: Grafo contraexemplo

```
1:  procedure ValidacaoConjecturaGerador
2:      n = 2
3:      repeat
4:          Seja T a menor coárvore com n vértices
5:          m ← Mgrau(T)
6:          if m > TetoDoIndice(T) + 1 then
7:              if Bcromatico(T) ≠ Cromatico(raiz(T)) then
8:                  if Bcromatico(T) > TetoDoIndice(T) + 1 then
9:                      stop
10:                 end if
11:             end if
12:         end if
13:         T ← ArvoreSeguinte(T)
14:     until T = null
15:     n ← n + 1
16: end function
```

Como a desigualdade 4.2 implica na veracidade da afirmação **Δ2**, a verificação executada na linha 6 aumenta consideravelmente a eficiência do algoritmo, por dois motivos:

- o cálculo do m-grau é notavelmente mais eficiente de ser obtido em comparação ao b-cromático;
- Conforme visto na tabela 4.4.2 é insignificante a quantidade de cografos conexos que não a satisfazem.

Nos pouquíssimos cografos que satisfazem a condicional na linha 6, verificamos na linha 7 se os números cromático e b-cromático coincidem, caso afirmativo então é claro que a afirmação deve ser satisfeita, pelo teorema 4.2.1. Caso negativo, por fim verificamos a afirmação na linha 8.

Executamos o algoritmo acima e não foi encontrado nenhum contra-exemplo para a afirmação **Δ2** para cografos conexos de até 21 vértices, contabilizando mais de 1 bilhão de grafos. Segue o resultado cuja demonstração foi feita computacionalmente.

Proposição 4.4.1. *A afirmação* **Δ2** *é satisfeita para cografos conexos de até 21 vértices.*

E propomos a seguinte conjectura:

AFIRMAÇÃO 4.4.1. *Todo cografo conexo G satisfaz $\chi_b(G) \leq \lceil \lambda(G) \rceil + 1$.*

Capítulo 5

Partição de arestas em cliques

Neste capítulo trataremos do problema de partição de arestas em cliques, que é um dos diversos problemas de partição e cobertura de arestas em grafos. Esses problemas foram introduzidos por Erdós, Goodman e Pósa [17], em 1966 e Lovász [40], 1968. Apresentamos abaixo a definição do problema em questão.

Definição 5.0.1. *Seja $G(V, E)$ um grafo, uma* ***cobertura*** *das arestas de G é uma família Ω formada por subconjuntos de E tal que cada subconjunto induz uma clique em G e $\bigcup_{C\in\Omega} = E$. Uma cobertura Ω é dita mínima se qualquer outra cobertura Ω' de G satisfaz $|\Omega'| \geq |\Omega|$, neste caso denotamos $|\Omega|$ por $cc(G)$.*

Exemplo 5.0.1. *No grafo ilustrado abaixo vemos que $cc(G) = 4$.*

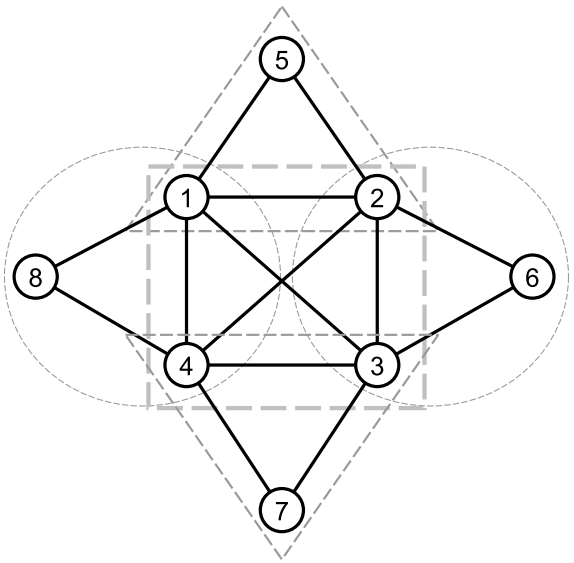

Figura 5.1: Exemplo de cobertura das arestas de G em cliques

Definição 5.0.2. *Seja $G(V, E)$ um grafo, uma* ***partição*** *das arestas de G é uma família Ω formada por subconjuntos de E tal que cada subconjunto induz uma clique em G e Ω é uma partição de E, ou seja, cada aresta de G pertence à exatamente um elemento de Ω. Uma partição Ω é dita mínima se qualquer outra partição Ω' de G satisfaz $|\Omega'| \geq |\Omega|$, neste caso denotamos $|\Omega|$ por $cp(G)$.*

Exemplo 5.0.2. *Abaixo ilustramos duas possíveis partições Ω e Ω' nesta ordem, das arestas do grafo em cliques. É fácil verificar que $cp(G) = |\Omega'| = 6$.*
$\Omega = \{E(G[1,2,3,4]), \{(1,8)\}, \{(4,8)\}, \{(4,7)\}\{(3,7)\}, \{(2,6)\}, \{(3,6)\}, \{(5,1)\}, \{(5,2)\}\}$.
$\Omega' = \{E(G[1,2,5]), E(G[1,4,6]), E(G[3,4,7]), E(G[2,3,6]), \{(1,3)\}, \{(1,4)\}\}$.

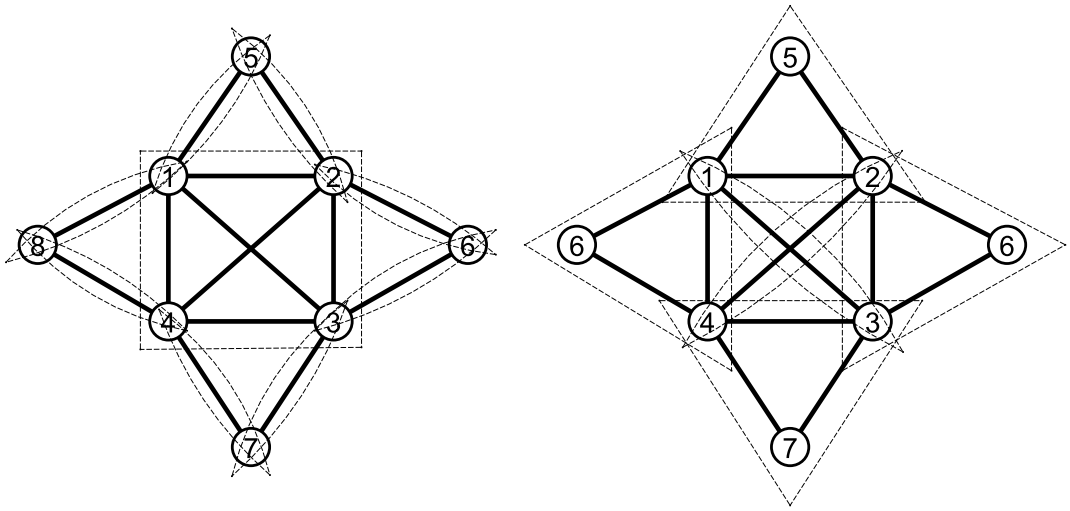

Figura 5.2: Possíveis partições de arestas em clique em um grafo

Vale ressaltar a diferença entre as Definições 5.0.1 e 5.0.2, onde na primeira as arestas podem ser repetidas dentre diferentes cliques, enquanto na segundo as cliques não apresentam interseção. Observe que uma partição das arestas em cliques também é uma cobertura, então como consequência imediata temos que, para todo grafo G vale $cc(G) \leq cp(G)$.

Uma variação do problema é particionar os **vértices** em cliques: Dado um grafo G, definimos o número clique de cobertura como sendo o tamanho da menor partição de $V(G)$ tal que cada elemento da partição induza uma clique em G. Porém tal problema é o complementar do problema da coloração de vértices, então é mais comum encontrar trabalhos com esta segunda abordagem. A saber, este é um dos 21 problemas listados por Richard Karp em [33].

Diversos trabalhos envolvendo cobertura de arestas encontram-se publicados tanto na área de cobertura quanto em partição do grafo por cliques em arestas. Na seção seguinte faremos uma breve revisão da literatura e mais adiante mostraremos nossas contribuições para esse problema.

5.1 Estudo da literatura sobre a partição das arestas em cliques

Nesta seção trataremos alguns resultados obtidos acerca do problema de partição das arestas em cliques. Isto será feito em duas subseções: a primeira tratará de resultados que fornecem propriedades, cotas e até o valor de $cp(G)$ de algumas classes; a segunda apresentará o estado da arte sob o ponto de vista de complexidade computacional.

5.1.1 Cotas e propriedades na literatura

O primeiro teorema a ser apresentado fornece uma cota superior para o número $cp(G)$ em função do número de vértices.

Teorema 5.1.1. *[17] Se G é um grafo com n vértices então existe uma partição das arestas em clique de G com tamanho máximo $\left\lfloor \frac{n^2}{4} \right\rfloor$, composta apenas por arestas e triângulos. Ou seja,*

$$cp(G) \leq \left\lfloor \frac{n^2}{4} \right\rfloor,$$

com igualdade mantida se, e somente se, $G \cong K_{\lfloor \frac{n}{2} \rfloor, \lceil \frac{n}{2} \rceil}$.

No próximo resultado podemos observar que ao efetuarmos ao particionarmos as arestas de uma clique de tamanho n temos apenas duas opções: particionar em uma única clique ou particionar em pelo menos n cliques.

Teorema 5.1.2. *[48] Se Ω é uma partição de K_n em cliques então uma, e apenas uma, das afirmações é válida:*

- $|\Omega| > n$;
- $|\Omega| = 1$, *onde* $\Omega = \{K_n\}$;
- $|\Omega| = n$, *onde* Ω *consiste de uma* $(n-1)-$*clique e* $(n-1)$ $2-$*cliques incidindo em um único vértice.*

- $|\Omega| = n$, *onde* Ω *consiste de* n *cópias de* K_{m+1} *para* $n = m^2 + m + 1$.

Corolário 5.1.1. *[48] Se* $n > 2$ *então* $cp(K_n - e) = n - 1$, *onde* e *representa uma aresta.*

Em 1977, Orlin [46] mostrou que o cálculo de $cp(G)$ e $cc(G)$ de um grafo linha é obtido diretamente do número de vértices de grau 2 do seu grafo raiz. Abaixo detalhamos as definições envolvidas no teorema que será descrito à seguir.

Definição 5.1.1. *Dado* $G(V, E)$ *definimos o* ***grafo linha*** $\ell(G)$ *como o grafo cujos vértices são as arestas de* G *e dois vértices são adjacentes em* $\ell(G)$ *se as respectivas arestas em* G *incidem em um vértice comum.*

Definição 5.1.2. *Um grafo* G *é um grafo linha se existe grafo* H *tal que* $\ell(H) \cong G$, *onde* H *é chamado* ***grafo linha-raiz*** *de* G.

Note que as arestas em $\ell(G)$ são induzidas por um par de arestas em G compartilhando um vértice, portanto cada vértice $v \in V(G)$ com $d(v) \geq 2$ gera uma clique $Q(v)$ em $\ell(G)$, cuja ordem é $d_G(v)$. Então o $\{Q(v); v \in V(G) e d_G(v) \geq 2\}$ é uma partição das arestas de $\ell(G)$ em cliques. Mais detalhes podem ser vistos em [46], onde temos o seguinte teorema.

Teorema 5.1.3. *[46] Seja* G *um grafo conexo não isomorfo à* K_3, *considere ainda* $V_2(G)$ *o conjunto de vértices de grau pelo menos 2 e* t *o número de triângulos com exatamente dois vértices de grau exatamente 2 em* G. *Então*

$$cp(\ell(G)) = |V_2(G)| \qquad e \qquad cc(\ell(G)) = |V_2(G)| - t.$$

Exemplo 5.1.1. *Na Figura 5.3 ilustramos um grafo* G *e abaixo o seu grafo linha com uma partição e uma cobertura das arestas em cliques, nesta ordem. Pelo Teorema* 5.1.3 *temos* $cp(\ell(G)) = 6$ *e* $cc(\ell(G)) = 5$. *Vale ressaltar que são grafos sem arestas duplas, a esboço é apenas para retratar que uma aresta foi escolhida em duas cliques da cobertura.*

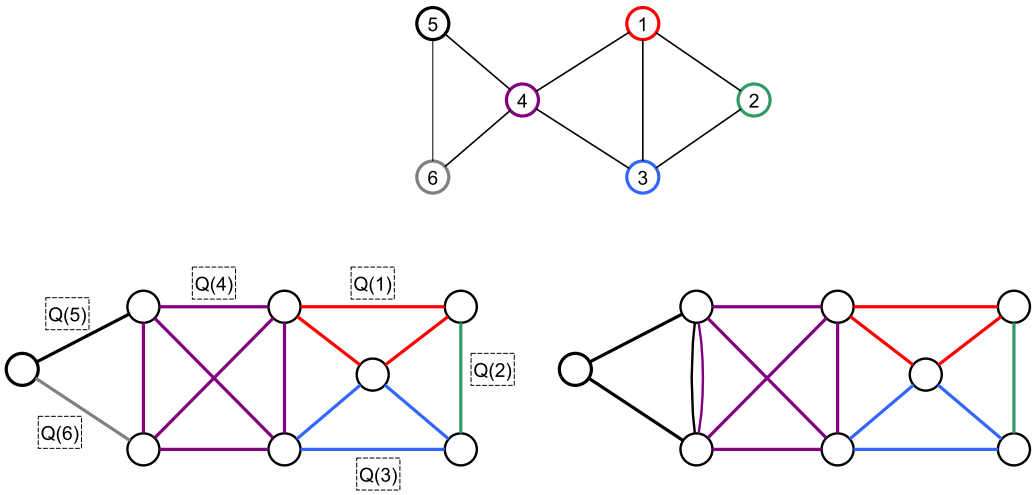

Figura 5.3: Grafo G e grafo $\ell(G)$ com partição e cobertura em cliques

Existem diversas generalizações da operação grafo linha, veja [3], destacaremos aqui a trabalhada nos artigos como [37] [34].

Definição 5.1.3. *Dado inteiro $k \geq 1$, **o grafo $k-$linha** $\ell_k(G)$ de um grafo tem seus vértices representando k-cliques de G e dois vértices são adjacentes em $\ell_k(G)$ se as cliques correpondentes se interceptam em uma $(k-1)$-clique.*

Note que para $k = 1$ temos $\ell_1(G) = K_n$ para todo grafo G de ordem n; para $k = 2$ temos $\ell_2(G) = \ell(G)$ a noção usual de grafo linha; para $k = 3$ obtemos o chamado grafo triangular linha. Um grafo é **triangular linha** se é o 3-linha de algum grafo. Em especial denotamos o operador ℓ_3 por $\mathcal{T}$, e podemos fazer referência simplesmente por grafo triangular. Estes foram estudados em diversos artigos, dentre eles [35], [2],[49].

Em especial, Pullman [49], obteve algumas propriedades relacionando grafo triângulo e a partição das arestas em cliques, conforme enunciado no corolário abaixo, cuja demonstração será apresentada na prova do Lema 5.2.1 na seção 5.2.1 como contribuição deste trabalho, pois foi obtida independentemente do artigo citado.

Corolário 5.1.2. *[49] Se G é K_4-livre, então $cp(G) = |E(G)| - 2\alpha(\ell_3(G))$, onde $\alpha(G)$ é o número de independência de G, isto é, o tamanho do maior conjunto estável do grafo.*

Exemplo 5.1.2. *Abaixo ilustramos G k_4-livre e seu grafo triângulo $\ell_3(G)$, nesta ordem. Pelo corolário obtemos*

$$cp(G) = |E(G)| - 2\alpha(\ell_3(G)) = 14 - 2 \cdot 4 = 6$$

.

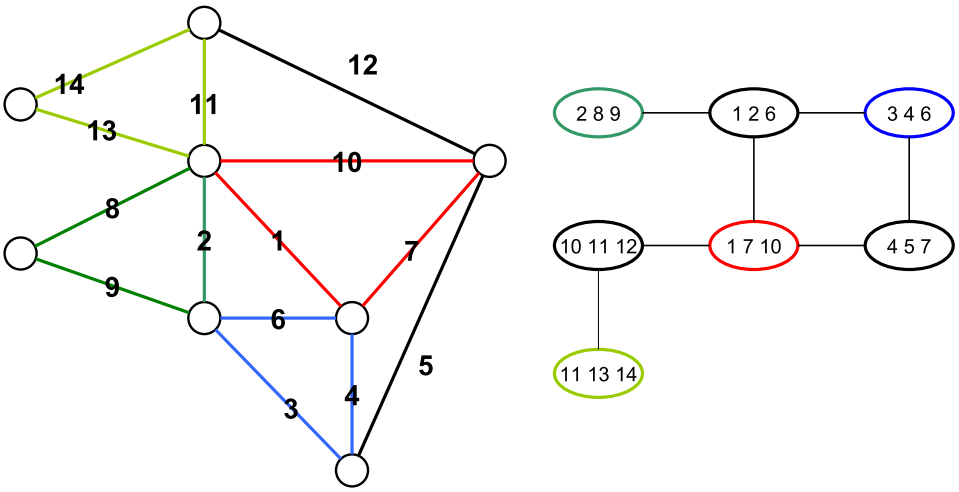

Figura 5.4: Grafo G k_4-livre com partição das arestas usando $\ell_3(G)$

Existem variações do problema de cobertura e partição de arestas, uma delas é obter o menor número de cliques maximais necessárias para particionar as arestas de um grafo, tal valor é denotado por $mcp(G)$. A saber uma clique é dita maximal se não está contida em nenhuma outra clique maior. Vale observar que todo grafo possui uma partição (e cobertura) de arestas por cliques, basta tomar cada aresta isoladamente. Contudo nem todo grafo possui uma partição por cliques maximais, o grafo *diamond* (obtido por $K_4 - e$) é um exemplo onde este problema não tem solução. Em geral, vale a relação $cc(G) \leq cp(G) \leq mcp(G)$, caso $mcp(G)$ exista.

Wallis e Ma provaram em [53] que se G é um grafo threshold então $cc(G) = cp(G) = mcp(G)$. A saber:

Definição 5.1.4. *Um* ***grafo threshold*** *é definido de forma recursiva:*

- *(1) Um vértice isolado é um threshold*
- *(2) Se G é threshold, então $G \cup \{v\}$ é threshold.*
- *(3) Se G é threshold, então $G \vee \{v\}$ é threshold.*

Seja G um grafo threshold. Para obter a partição mínima das arestas em clique de G basta tomar as cliques maximais do grafo, ou seja, a partição é dada pela clique máxima e cópias do K_2, sabendo que tal clique máxima é construída pela operação (3) da definição da classe. Logo, obter a menor partição das arestas em cliques de um threshold é feito em tempo polinomial.

Diversas outras propriedades podem ser vistas nos artigos citados acima, nos surveys [48] [45] e outros trabalhos, tratando classes como grafos $k-$regulares, grafos $K_n \backslash K_m$, $\overline{P_n}$, $\overline{C_n}$, grafos limitados pelo grau máximo, dentre outros.

5.1.2 Complexidade ECP na literatura

Nesta subseção estudaremos os resultados existentes na literatura sob o ponto de vista de complexidade computacional, apresentando o estado da arte deste problema. Inicialmente formalizaremos os problemas em questão:

Problema 5.1.1. ***(ECP)***
INSTÂNCIA: Grafo G e inteiro $k \geq 0$.
QUESTÃO: Vale $cp(G) \leq k$?

Problema 5.1.2. ***(ECC)***
INSTÂNCIA: Grafo G e inteiro $k \geq 0$.
QUESTÃO: Vale $cc(G) \leq k$?

Em 1977, Orlin [46] provou que os problemas ECP e ECC são $\mathcal{NP}$-completo para grafos gerais. Já em 1988, Ma, Wallis e Wu [41] provaram que ECP é $\mathcal{NP}$-completo para classe de grafos livres de K_4 e grafos cordais, porém é polinomial se restrito à interseção destas classes.

Posteriormente em 1991 foi provado por Wallis e Wu [56] que o problema permanece $\mathcal{NP}$-completo para os grafos *split*, ainda que os vértices do conjunto estável tenham grau 2. Note que a classe dos *split* (ou $(1,1)$) é uma subclasse dos grafos cordais.

Sob o ponto de vista de grau máximo, em 1981, Pullman [49] provou que os problemas ECP e ECC são polinomiais para grafos com $\Delta(G) \leq 4$, apresentando algoritmo para a obtenção dos valores e da partição/cobertura. Em 1992, Hoover [25] apresentou um algoritmo polinomial para o problema ECC para grafos com $\Delta(G) \leq 5$ e ainda garantiu que o problema é $\mathcal{NP}$-completo para $\Delta(G) \leq 6$, quanto ao ECP neste mesmo artigo provou que o problema é $\mathcal{NP}$-completo para $\Delta(G) \leq 5$. Ainda nas classes definidas por grau, Fleischer e Wu [19] provaram que o problema é polinomial para grafos cúbicos, isto é, todos vértices possuem grau 3.

Abaixo listamos um resumo das complexidades citadas anteriormente para o problema ECP em algumas classes de grafos.

- Grafos gerais: $\mathcal{NP}$-completo [46];

- Cordais: $\mathcal{NP}$-completo [41];
- K_4-livre: $\mathcal{NP}$-completo [41];
- Cordais e K_4-livre: polinomial [41];
- Split: $\mathcal{NP}$-completo [56];
- Threshold: polinomial [53]
- $\Delta(G) \leq 4$: polinomial [49];
- $\Delta(G) \leq 5$: $\mathcal{NP}$-completo [25].
- 3-regular: polinomial [19]
- Grafo Linha: polinomial [46]

Além dos artigos citados nas subseções 5.1.1 e 5.1.2 também vale citar as seguintes teses: *" The NP-completeness of some edge-partitioning problem "* [10] de 2002 que estuda a complexidade de ECP e ECC em grafos com grau máximo fixo; *" Maximal-clique partitions "* [55] que aborda o problema de partição das arestas em cliques maximais em grafos linha e a existência de grafos que satisfazem $cc(G) = cp(G) < mcp(G)$ e o caso $cc(G) < cp(G) < mcp(G)$.

5.2 ECP em grafos (k,l)

Focaremos nosso estudo no problema de ECP sob o ponto de vista de grafos (k, l). Abaixo listaremos os resultados de complexidade imediatos.

Observação 5.2.1. *O problema de ECP em grafos* $(1, 0)$ *ou* $(0, 1)$ *é trivial, portanto apresenta solução polinomial no tamanho do grafo.*

Observação 5.2.2. *Os grafos* $(2, 0)$ *formam a classe dos grafos bipartidos, onde a partição de arestas em cliques é dada por cópias de* K_2*, então* $cp(G) = |E(G)|$.

Apresentaremos agora a teoria e os resultados desenvolvidos neste trabalho. Inicialmente vale observar que conforme mencionado acima, Wallis provou que ECP é um problema $\mathcal{NP}$-completo para a classe dos grafos $(1, 1)$, portanto podemos garantir que o problema é $\mathcal{NP}$-completo para grafos (k, ℓ), com $k, \ell \geq 1$ fixados, uma vez que trata-se

de uma superclasse, conforme visto pelo Fato 2.4.1. O que direciona nosso estudo para as classes de grafos $(0, \ell)$ e $(k, 0)$ para $k \geq 2$ e $\ell \geq 3$

Um grafo (0,2) é aquele cujo conjunto de vértices pode ser particionado em duas cliques, chamado de grafos co-bipartidos.

Exemplo 5.2.1. *O grafo ilustrado abaixo é (0,2)*

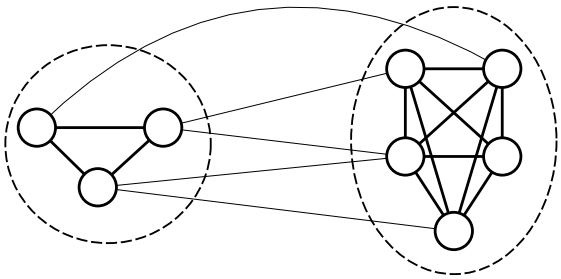

Figura 5.5: Exemplo grafo (0,2)

A primeira contribuição deste trabalho para o problema de ECP é a proposição seguinte, que fornece a complexidade de tempo para um grafo $(0, 2)$.

Proposição 5.2.1. *O problema de partição de arestas em clique em grafos* $(0, 2)$ *é NP-completo.*

Demonstração. Seja $G(V, E)$ um grafo split com n vértices, digamos que V é particionado em dois conjuntos $V = A \cup B$ tais que A é um conjunto independente e B uma clique. Considere $N = \frac{n(n-1)}{2}$ e seja $G'(V', E')$ onde V' é particionado nos conjuntos A' e B' tais que $A' = A \cup \{u_1, \ldots, u_N\}$ e $B' = B$, as arestas por sua vez formam o conjunto $E' = E \cup \{(x, y); x, y \in A'\}$, conforme ilustrado na Figura 5.6.

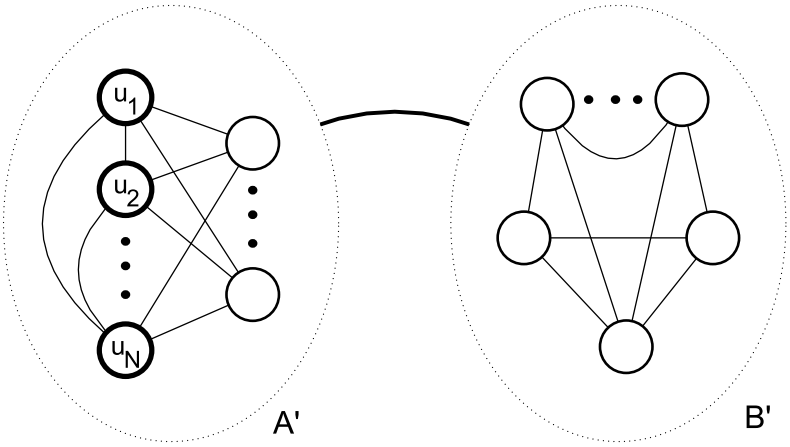

Figura 5.6: Grafo G' construído a partir do split G

Observe que G' é um grafo $(0,2)$ com $n+N$ vértices, onde N é o número de arestas de uma n-clique. Sejam Ω e Ω' as famílias com menor número de cliques que particionam as arestas de G e G', nesta ordem. Denotaremos o conjunto das arestas de A' por C_1.

Se $C_1 \notin \Omega'$ então pelo Teorema 5.1.2 vale

$$|\Omega'| \geq N + |A| = \frac{n(n-1)}{2} + |A| \geq cp(G) + 1,$$

Onde a última desigualdade é obtida pelo fato de Ω não ser formada apenas por arestas isoladas, ou seja, $cp(G) = |\Omega| \leq \frac{n(n-1)}{2}$, uma vez que G possui ao menos um triângulo.

Se $C_1 \in \Omega'$ então resta apenas particionar as arestas de G, cujo mínimo é dado por $\Omega' = \Omega \cup C_1$, ou seja, $cp(G') = cp(G) + 1$.

Logo, dado um grafo G split construímos G' grafo $(0,2)$ tal que $cp(G') = cp(G) + 1$, construído através de uma redução polinomial. Como o problema ECP é NP-completo para grafos split (provado em [56], então é NP-completo para grafos $(2,0)$. □

Naturalmente a complexidade se estende para as superclasses do $(0,2)$, pelo Fato 2.4.1, conforme enunciamos abaixo:

Observação 5.2.3. *O problema de partição de arestas em clique em grafos $(0,\ell)$ é NP-completo para $\ell \geq 2$.*

5.2.1 ECP em grafos 3-colorível

Estudaremos agora a classe dos grafos $(3,0)$ ou também chamados de 3-coloríveis, pois tratam-se dos grafos com número cromático $\chi(G) = 3$.

Fato 5.2.1. *Seja G um grafo, $\{a,b,c\} \in E(G)$ um triângulo e Ω uma partição das arestas em cliques de G. Se $\{a\},\{b\},\{c\} \subset \Omega$ então Ω não é mínima.*

De fato, $\Omega' = \Omega - \{\{a\}\{b\},\{c\}\} \cup \{a,b,c\}$ é uma partição das arestas de G com cardinalidade menor que $|\Omega|$, portanto esta não é mínima.

Para provar a complexidade do problema de ECP em grafos 3−coloríveis, provamos o lema a seguir que relaciona este com o problema de obter o maior número de triângulos disjuntos em arestas de um grafos, definido abaixo.

Problema 5.2.1. *(**MTED** - maximum triangle edge-disjoint)*
INSTÂNCIA: Grafo G e inteiro $k \geq 0$.
QUESTÃO: Existe família $\mathcal{F}$ de triângulos disjuntos em arestas tal que $|\mathcal{F}| \geq k$?

Daqui em diante, a fim de simplificar a escrita,chamaremos de " triângulos adjacentes " aqueles que possuem uma aresta em comum. Caso seja necessário citarmos interseção em vértices, deixaremos explícito no texto.

Lema 5.2.1. *Se G é K_4-livre, então os Problemas ECP e MTED são equivalentes.*

Demonstração. Seja G um grafo K_4-livre com n vértices e m arestas, observe que qualquer partição do conjunto de arestas em cliques é formada apenas por triângulos e arestas isoladas.

Seja Ω a menor partição das arestas de G em cliques, formada por p triângulos não adjacentes e q arestas isoladas. Se removermos de G as arestas que compõem os p triângulos não adjacentes obtemos o grafo G' livre de K_3, pois caso contrário teria uma contradição com o Fato 5.2.1.

Como Ω é uma partição das arestas de G temos que $m = 3p + q$, ou de forma equivalente,$q = m - 3p$. Então o problema de obter a *ECP* em G (minimizar q) é equivalente à obter o maior número de triângulos não adjacentes do grafo (maximizar p). Note que $|\Omega| = p + q = p + (m - 3p) = m - 2p$. □

Observe que a prova do lema acima contém uma demonstração para o Corolário 5.1.2.

Para provar a complexidade do problema de *ECP* em grafos $(3, 0)$ faremos a redução polinomial do clássico problema de cobertura exata, restrito ao caso de 3 elementos, conforme descrito abaixo:

Problema 5.2.2. *(**E3C** - exact 3 cover)*
INSTÂNCIA: (X, ψ), onde X é um conjunto com $3q$ elementos e ψ família de subconjuntos de X, tal que cada elemento possui cardinalidade 3.
QUESTÃO: Existe uma subfamília $\psi' \subset \psi$ com q elementos que é uma cobertura exata para X? Isto é, cada elemento $x \in X$ pertence à exatamente um subconjunto de ψ'.

Note que o termo "cobertura exata" deve ser interpretado exatamente como a definição de partição, onde ψ' é formado por subconjuntos disjuntos que unidos resultam em X. Para uma melhor compreensão do problema E3C, observe o seguinte exemplo:

Exemplo 5.2.2. *Considere o conjunto $X = \{1, 2, 3, 4, 5, 6\}$ e considere o problema E3C com as seguintes instâncias:*

1. *DADOS: X e $\psi = \{S_i; i = 1, 2, 3, 4\}$, onde $S_1 = \{1, 2, 4\}$, $S_2 = \{2, 5, 6\}$, $S_3 = \{3, 5, 6\}$ e $S_4 = \{4, 5, 6\}$.*
   Possui solução para o problema E3C, tomando $\psi' = \{S_1, S_3\}$.

2. *DADOS: X e $\psi = \{S_1, S_2, S_3\}$, onde $S_1 = \{1, 2, 3\}$, $S_2 = \{3, 4, 5\}$ e $S_3 = \{3, 4, 6\}$.*
   Não possui solução para o problema E3C.

O problema E3C é NP-completo, conforme provado em [21]. Faremos então a redução polinomial deste problema para provar a complexidade do problema *E3C* para grafos $(3, 0)$.

Teorema 5.2.1. *O ECP é NP-completo para grafos 3-colorível.*

Demonstração. Inicialmente é fácil ver que ECP restrito à grafos 3-colorível está em $\mathcal{NP}$, pois dada uma partição das arestas podemos verificar em tempo polinomial (no tamanho da entrada) que esta de fato é uma solução para a resposta SIM do problema. Faremos agora a construção da redução polinomial.

Observe que o lema 5.2.1 garante que o problema de partição de arestas em clique em grafos é equivalente ao problema de obter o número máximo de triângulos não adjacentes, para grafos 3-colorível. Então a prova de complexidade será feita fazendo a redução do problema E3C para o problema de obter o maior número de triângulos não adjacentes.

Seja a instância $X = \{x_i; i = 1, 2, \ldots, 3q\}$ e $\psi = \{S_i; i = 1, 2, \ldots, p\}$ onde $S_i = \{x_{i,1}, x_{i,2}, x_{i,3}\} \subset X$, $i = 1, 2, \ldots, p$.

Faremos a construção do grafo G da seguinte forma:

- Para cada $x \in X$ associamos dois vértices x e x'.
- Para cada $i = 1, \ldots, p$ associamos S_i à 7 vértices $s_{i,j}$, $j = 1, \ldots, 7$.
- Para cada $i = 1, \ldots, p$ construímos o grafo G_i conforme ilustrado na Figura 5.7.

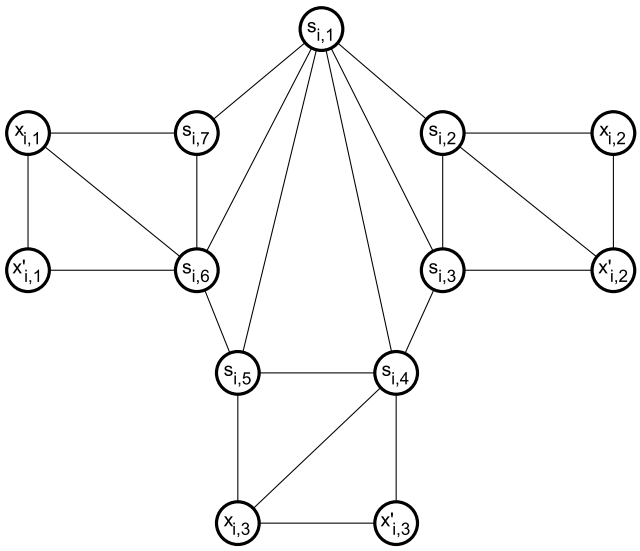

Figura 5.7: Construção do Grafo G_i referente à prova do teorema 5.2.1

Então construímos G tal que $V(G) = X \cup X' \cup \{s_{i,j}; 1 \leq i \leq p \text{ e } 1 \leq j \leq 7\}$, onde $X' = \{x'; x \in X\}$ e conjunto de arestas $E(G) = \bigcup_{i=1}^{p} E(G_i)$.

Vale observar que os grafos G_i não são disjuntos em vértices nem arestas, uma vez que se $x \in S_i \cap S_j$ para $i \neq j$ então $x, x' \in V(G_i) \cap V(G_j)$ e $xx' \in E(G_i) \cap E(G_j)$.

Vamos agora provar que (X, ψ) tem uma cobertura exata se, e somente se, G possui $q + 5p$ triângulos disjuntos em arestas.

Suponha $\psi' \subset \psi$ uma cobertura exata para X, neste caso temos $|\psi'| = q \geq p$ e digamos $\psi' = \{S_1, S_2, \ldots, S_q\}$.

Para $i = 1, 2, \ldots, q$, os grafos G_i são disjuntos dois a dois (em vértices e arestas), pois ψ' é uma cobertura exata de X, então cada G_i possui 6 triângulos disjuntos em arestas (veja a Figura 5.8), neste caso todas as arestas do tipo xx' estão cobertas por

algum triângulo, uma vez que ψ' é uma cobertura exata para X. Os 6 triângulos para cada G_i são: $\{s_{i,1}, s_{i,6}, s_{i,7}\}$, $\{s_{i,1}, s_{i,4}, s_{i,5}\}$, $\{s_{i,1}, s_{i,2}, s_{i,3}\}$, $\{x_{i,1}, x'_{i,1}, s_{i,7}\}$, $\{x_{i,2}, x'_{i,2}, s_{i,2}\}$ e $\{x_{i,3}, x'_{i,3}, s_{i,5}\}$. Totalizando $6q$ triângulos.

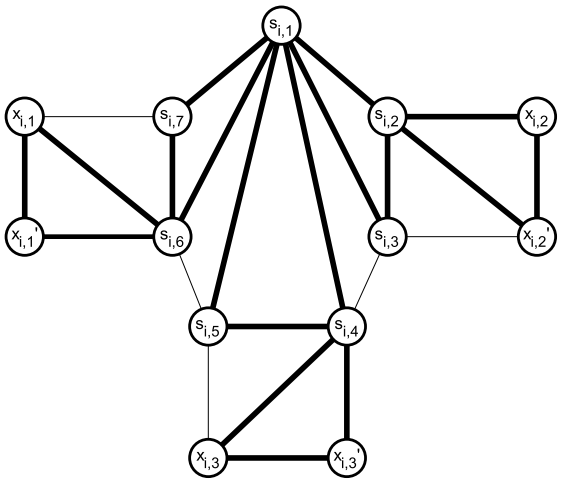

Figura 5.8: Triângulos em G_i referente à prova do teorema 5.2.1

Para $i = q+1, q+2, \ldots, p$, caso exista, cada G_i não possui 6 triângulos disjuntos, pois as arestas xx' já pertencem à triângulos do caso anterior, portanto cada G_i possui 5 triângulos disjuntos (veja a Figura 5.9), a saber: $\{s_{i,1}, s_{i,5}, s_{i,6}\}$, $\{s_{i,1}, s_{i,3}, s_{i,4}\}$, $\{s_{i,7}, s_{i,6}, x'_{i,1}\}$, $\{s_{i,4}, s_{i,5}, x'_{i,3}\}$, $\{s_{i,2}, s_{i,3}, x'_{i,2}\}$. Totalizando $5(p-q)$ triângulos.

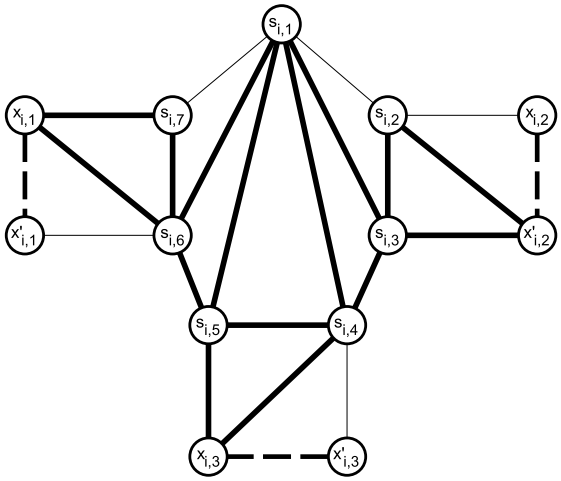

Figura 5.9: Triângulos em G_i referente à prova do teorema 5.2.1

Então G possui ao todo $6q + 5(p-q) = q + 5p$.

Reciprocamente, suponha que G possui $q+5p$ triângulos. Pelo raciocínio desenvolvido

acima temos que cada G_i possui no máximo 6 ou 5 triângulos disjuntos em arestas, neste caso existem q grafos G_i com 6 triângulos e $(p-q)$ grafos G_i com 5 triângulos. Note que a única forma de obter 6 triângulos em um grafo G_i é conforme descrito no caso acima e visto na Figura 5.8, com essa configuração cada G_i com 6 triângulos possui duas características: cobre exatamente três elementos de X, a saber $x_{i,1}, x_{i,2}$ e $x_{i,3}$; está associado à S_i, apenas. Como existem q grafos com essa configuração, cobrimos os $3q$ elementos de X, uma vez que os triângulos são disjuntos e portanto não há repetição de elementos (arestas). Logo X possui uma cobertura exata, dada pelos G_i com 6 triângulos.

Vamos agora provar que $q + 5p$ é o máximo de triângulos disjuntos em G, atingido apenas no caso em que (X, ψ) possui uma cobertura exata. De fato, digamos que existem a grafos G_i com 6 triângulos disjuntos e b grafos G_i com 5 triângulos disjuntos, então G possui $6a+5b$ triângulos disjuntos, onde $a+b = p = |\psi|$. O máximo da expressão $6a+5b$ ocorre quanto a atinge seu máximo q, pois $a > q$ geraria uma contradição com o fato $|X| = 3q$. Logo temos $b = p - q$ e obtemos o desejado.

Resta apenas provar que o grafo G é 3-colorível. De fato, o grafo G é construído pelos grafos G_i após identificação dos vértices x_i e x'_i para $i = 1, \ldots, 3q$. Note que cada G_i pode ser colorido com apenas 3 cores (veja a Figura 5.10), de forma que $cor(x_i) \in \{1, 2\}$ e $cor(x'_i) \in \{1, 2\}$, para cada $i = 1, \ldots, 3q$. O que garante que as identificações dos vértices podem ser feitas sem alterar a coloração de cada G_i. Logo G é 3-colorível.

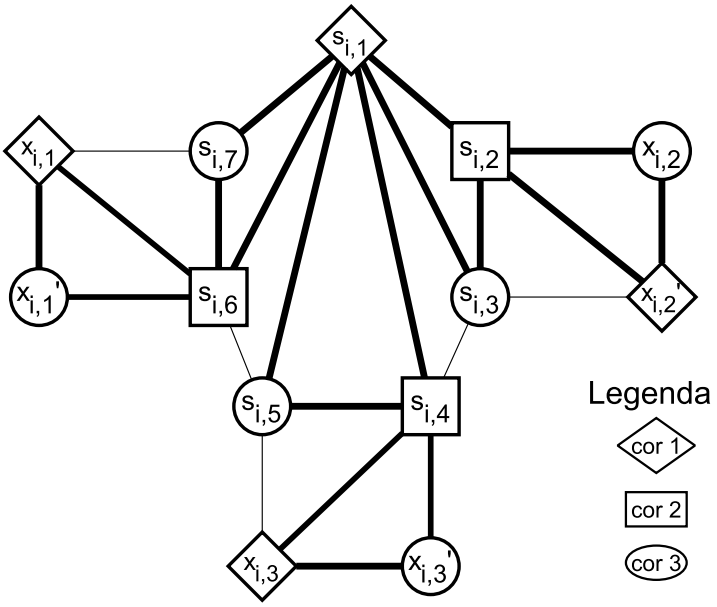

Figura 5.10: Coloração em G_i referente à prova do teorema 5.2.1

Como a construção de G a partir de (X, ψ) pode ser feita em tempo polinomial do

tamanho da instância e $E3C$ é um problema $\mathcal{NP}$-completo então o problema é $\mathcal{NP}$-completo para grafos 3-colorível. □

Observe o exemplo a seguir.

Exemplo 5.2.3. *Considere o problema E3C cuja instância é* $X = \{1,2,3,4,5,6\}$ *e* $\psi = \{S_1, S_2, S_3\}$, *onde* $S_1 = \{1,2,3\}$, $S_2 = \{1,5,6\}$ *e* $S_3 = \{4,5,6\}$. *Na Figura 5.11 construímos* $G = G_1 \cup G_2 \cup G_3$ *conforme descrito na demonstração do Teorema 5.2.1. Observe que* $q = 2$, $p = 3$ *e* G *possui* $q + 5p = 17$ *triângulos não adjacentes.*

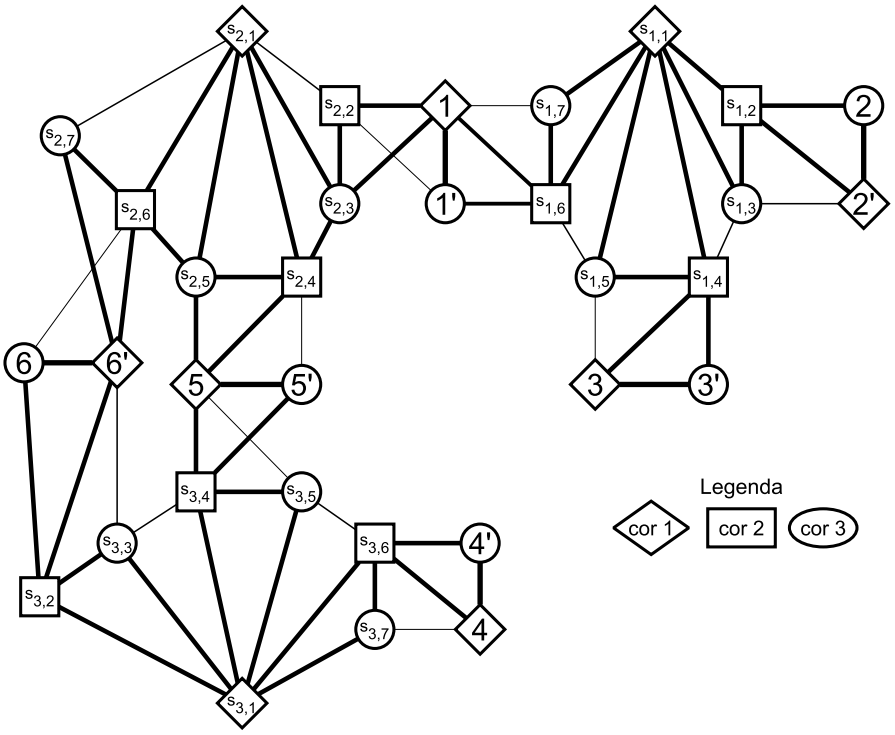

Figura 5.11: Exemplo da redução feita na demonstração do Teorema 5.2.1

Abaixo ilustramos a classe dos grafos (k,l) de acordo com a complexidade do problema ECP. Os nós marcados com $*$ indicam contribuição deste trabalho.

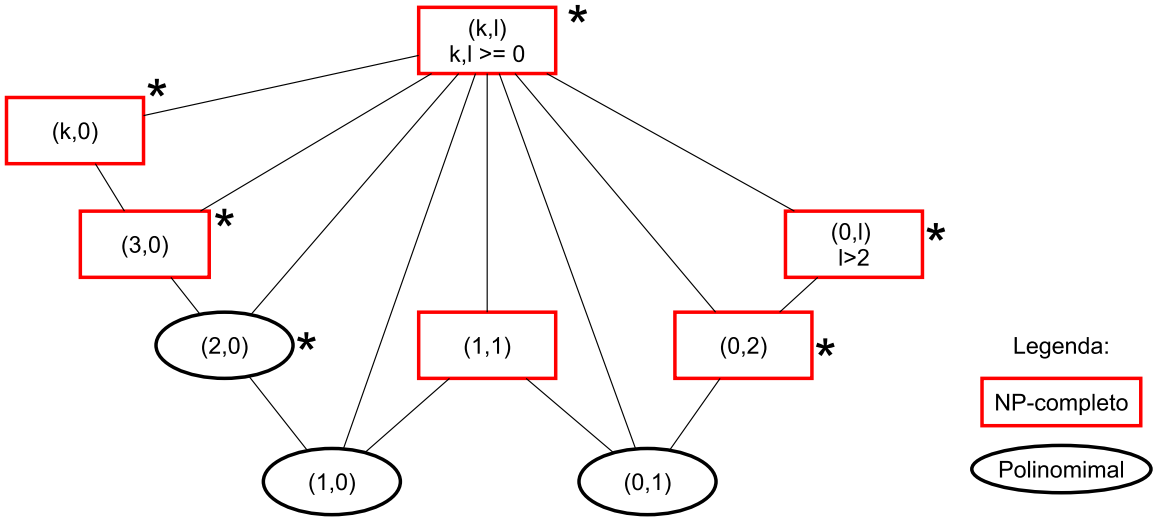

Figura 5.12: Esquema de Subclasses (k, l) de acordo com a complexidade de ECP

Observe que a classe dos grafos 3-colorível é uma subclasse dos grafos livres de K_4, então vale ressaltar que o nosso Teorema 5.2.1 é um refinamento do Teorema obtido por Ma, Wallis e Wu [41] em 1988, onde provaram que ECP é $\mathcal{NP}$-completo para classe de grafos livres de K_4.

Por uma aplicação direta do Corolário 5.1.2 e Teorema 5.2.1 provamos a complexidade do problema da obtenção do número independente máximo em grafos triangulares linha.

Problema 5.2.3. *(**MIS**- maximum independent set)*
INSTÂNCIA: Grafo G e inteiro k
QUESTÃO: Vale $\alpha(G) \leq k$?

Corolário 5.2.1. *O problema MIS é $\mathcal{NP}$-completo em grafos triângulos $\ell_3(G)$, ainda que seu grafo raiz G seja 3-colorível.*

5.3 Subclasses com ECP polinomial

Nesta seção apresentaremos subclasses onde o problema ECP deixa de ser $\mathcal{NP}$-completo e se torna polinomial.

5.3.1 Grafos split-indiferença

Um grafo $G(V, E)$ é um **grafo indiferença** se existe uma função $f : V \to \mathbb{R}$ e um número real k tal que dois vértices $u, v \in V$ são adjacentes se $|f(u) - f(v)| \leq k$. Vale observar que

a classe dos grafos indiferença é uma subclasse dos grafos cordais. Um grafo que é split e de indifença é dito grafo **split-indiferença**, classe introduzida em [47] onde é provada uma caracterização da classe, conforme enunciamos no Teorema abaixo. Para uma leitura mais detalhada sobre grafos cordais e suas subclasses sugerimos [42].

Teorema 5.3.1. *Seja G um grafo conexo, então G é split-indiferença se, e somente se:*

1. *G é um grafo completo; ou*
2. *G possui duas cliques maximais Q_1 e Q_2, tais que $|Q_1 - Q_2| = 1$; ou*
3. *G possui três cliques maximais Q_1, Q_2 e Q_3, tais que:*
   - $|Q_1 - Q_2| = |Q_3 - Q_2| = 1$
   - $Q_1 \cap Q_3 = \emptyset$ *ou* $Q_1 \cup Q_3 = V$.

A Figura 5.13 esquematiza os casos do Teorema acima, onde

- A ilustração 3-a se refere ao caso 3 onde

$$|Q_1 - Q_2| = |Q_3 - Q_2| = 1, Q_1 \cap Q_3 = \emptyset \text{ e } Q_1 \cup Q_3 \neq V;$$

- A ilustração 3-b se refere ao caso 3 onde

$$|Q_1 - Q_2| = |Q_3 - Q_2| = 1, Q_1 \cap Q_3 \neq \emptyset \text{ e } Q_1 \cup Q_3 = V.$$

- A ilustração 3-c se refere ao caso 3 onde

$$|Q_1 - Q_2| = |Q_3 - Q_2| = 1, Q_1 \cap Q_3 = \emptyset \text{ e } Q_1 \cup Q_3 = V;$$

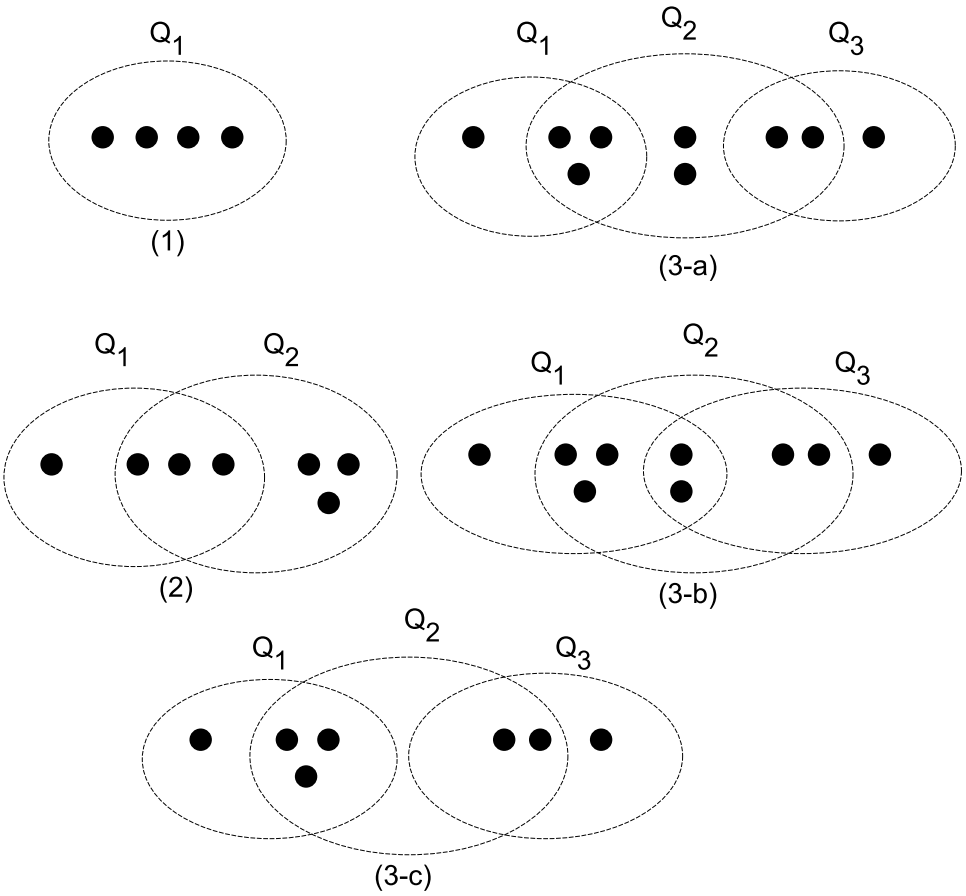

Figura 5.13: Esquema dos casos do Teorema 5.3.1

No próximo resultado provamos a complexidade do Problema ECP nesta classe de grafos.

Proposição 5.3.1. *O problema ECP é polinomial para a classe dos grafos split-indiferença conexos. Além disso dado G nesta classe, valem:*

1. *Se G possui exatamente uma clique maximal, então $cp(G) = 1$;*
2. *Se G possui exatamente duas cliques maximais Q_1 e Q_2, tais que $|Q_1 - Q_2| = 1$, então $cp(G) = 1 + |Q_1 \cap Q_2|$;*
3. *Se G possui exatamente três cliques maximais Q_1, Q_2 e Q_3, tais que $|Q_1 - Q_2| = |Q_3 - Q_2| = 1$ e $Q_1 \cap Q_3 = \emptyset$ então $cp(G) = 1 + |Q_1 \cap Q_2| + |Q_2 \cap Q_3|$;*
4. *Se G possui exatamente três cliques maximais Q_1, Q_2 e Q_3, tais que $|Q_1 - Q_2| = |Q_3 - Q_2| = 1$, $Q_1 \cup Q_3 = V(G)$ e $Q_1 \cap Q_3 \neq \emptyset$ então*

   $cp(G) = min\{1 + |Q_1 \cap Q_2| + |Q_2 \cap Q_3|, 2 + |Q_3 - Q_1| \cdot (|Q_1 \cap Q_3| - 1) + (|Q_1 - Q_3| - 1)(|Q_3 - Q_1| - 1), 2 + |Q_1 - Q_3| \cdot (|Q_1 \cap Q_3| - 1) + (|Q_1 - Q_3| - 1)(|Q_3 - Q_1| - 1)\}$

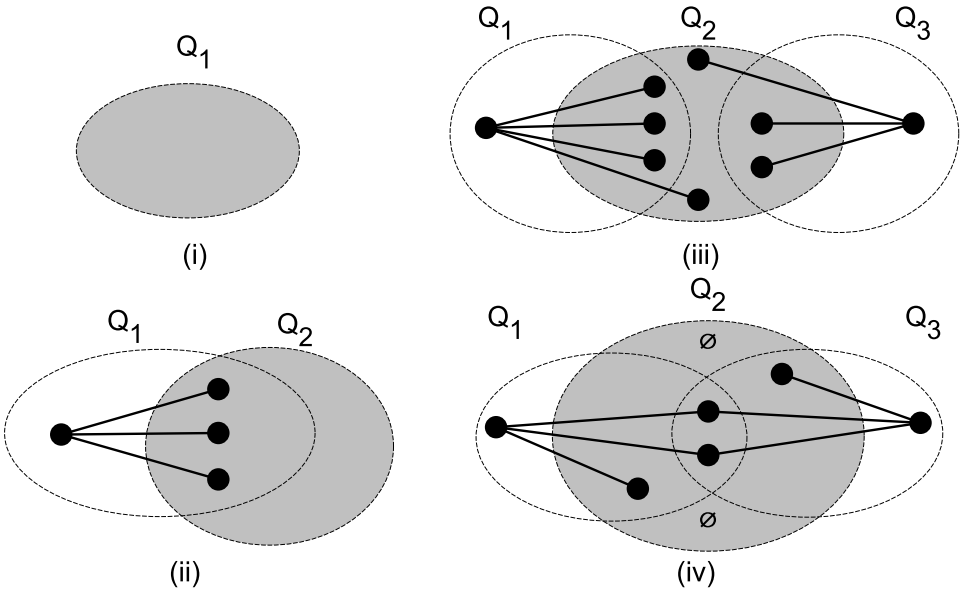

Figura 5.14: Esquema dos casos do Teorema 5.3.1

Demonstração. Considere os casos:

1. Trivial.

2. G possui duas cliques maximais Q_1 e Q_2 tais que $|Q_1 - Q_2| = 1$. Então $G[E(G) - E(Q_2)] \cong (K_1 \vee K_{|Q_1 \cap Q_2|})$. Seja Ω a menor partição das arestas de G em cliques,

   Se $Q_2 \in \Omega$ então a partição é formada por Q_2 e as arestas isoladas de $K_1 \vee K_{|Q_1 \cap Q_2|}$, ou seja, $|\Omega| = 1 + |Q_1 \cap Q_2|$. Veja a Figura 5.14(ii).

   Se $Q_2 \notin \Omega$, então pelo Teorema 5.1.2 vale $|\Omega| \geq |Q_2| \geq |Q_1 \cap Q_2| + 1$.

   Logo o mínimo ocorre para $Q_2 \in \Omega$, onde $cp(G) = 1 + |Q_1 \cap Q_2|$.

3. Seja G com três cliques maximais Q_1, Q_2 e Q_3, tais que $|Q_1 - Q_2| = |Q_3 - Q_2| = 1$ e $Q_1 \cap Q_3 = \emptyset$. Neste caso $G[E(G) - E(Q_2)]$ é livre de triângulos.

   Seja Ω a menor partição das arestas de G em cliques.

   Se $Q_2 \in \Omega$ então a partição é formada por Q_2 e as arestas isoladas de $G[E(G) - E(Q_2)]$, ou seja, $|\Omega| = 1 + |Q_1 \cap Q_2| + |Q_2 \cap Q_3|$. Veja a Figura 5.14(iii).

   Por outro lado se $Q_2 \notin \Omega$ então pelo Teorema 5.1.2 vale $|\Omega| \geq |Q_2| + 1 \geq |Q_1 \cap Q_2| + |Q_2 \cap Q_3| + 1$, pois $Q_1 \cap Q_3 = \emptyset$. A primeira desigualdade tem o acréscimo de uma unidade pelo fato de G possuir arestas com extremos em $V - Q_2$.

   Logo o mínimo ocorre com $Q_2 \in \Omega$, donde $cp(G) = 1 + |Q_1 \cap Q_2| + |Q_2 \cap Q_3|$.

4. Seja G com três cliques maximais Q_1, Q_2 e Q_3, tais que $|Q_1 - Q_2| = |Q_3 - Q_2| = 1$, $Q_1 \cup Q_3 = V(G)$ e $Q_1 \cap Q_3 \neq \emptyset$. A fim de não sobrecarregar a escrita da prova considere:

   $a := Q_1 \cap Q_2 - Q_3$, $b := |Q_1 \cap Q_3|$, $c := Q_3 \cap Q_2 - Q_1$, $v_1 := Q_1 - Q_2$ e $v_3 := Q_3 - Q_2$,

   conforme ilustrado na Figura 5.15.

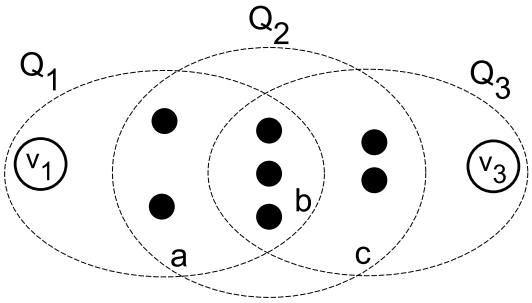

Figura 5.15: Representação do CASO 4 da prova do Teorema 5.3.1

Seja Ω a menor partição das arestas de G em cliques.

Considere os quatro casos a seguir:

CASO A: Se $Q_2 \in \Omega$ então a partição é formada por Q_2 e as arestas isoladas de $G[E - E(Q_2)]$, ou seja,

$$|\Omega| = 1 + (a + b) + (c + b) = 1 + |Q_1 \cap Q_2| + |Q_3 \cap Q_2|. \tag{5.1}$$

CASO B: Se $Q_1 \in \Omega$ então a partição é formada por (veja a Figura 5.16):

- Q_1: 1 clique;
- Clique $(Q_3 - Q_1) \cup u$, onde $u \in Q_1 \cap Q_3$: 1 clique;
- As arestas entre $Q_3 - Q_1$ e $(Q_1 \cap Q_3 - \{u\})$: $(c + 1)(b - 1)$ cliques;
- As arestas entre $Q_3 - Q_1 - \{v_3\}$ e $Q_1 - Q_3 - \{v_1\}$: ac cliques.

Então

$$|\Omega| = 1+1+(c+1)(b-1)+ac = 1+1+|Q_3-Q_1|\cdot(|Q_1\cap Q_3|-1)+(|Q_1-Q_3|-1)(|Q_3-Q_1|-1)$$

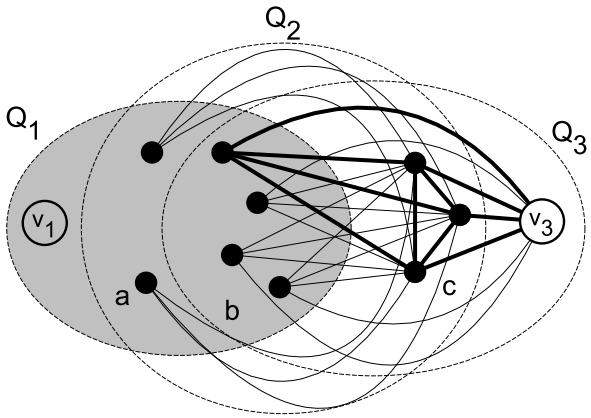

Figura 5.16: Caso $Q_1 \in \Omega$ na prova do Teorema 5.3.1

CASO C: Se $Q_3 \in \Omega$ temos caso análogo ao anterior, por simetria:

$$|\Omega| = 1+1+(a+1)(b-1)+ac = 1+1+|Q_1-Q_3|\cdot(|Q_1\cap Q_3|-1)+(|Q_1-Q_3|-1)(|Q_3-Q_1|-1).$$

CASO D: Se $Q_1, Q_2, Q_3 \notin \Omega$.

Pelo Teorema 5.1.2 vale $|\Omega| \geq |Q_2| = a+b+c \quad (*)$.

Para cada clique $C \in \Omega$ que contenha $\{v_1, u_1, \ldots, u_k\}$, onde $u_i \in Q_1 \cap Q_3$, as k arestas v_3u_i, $1 \leq i \leq k$, devem ser tomadas separadamente em Ω, pois as arestas do tipo u_iu_j estão na clique C; analogamente para as cliques que contém v_3 e outros vértices da interseção. Portanto b arestas do tipo v_1u_j ou v_3u_j devem ser tomadas isoladamente em Ω. A contagem destas arestas não fazem parte da contagem $(*)$, pois são arestas que não pertencem à $G[Q_2]$, isto é, $|\Omega| \geq |Q_2| + \mathbf{b} = a+2b+c$. Observe que caso não exista a clique C suposta acima, então as arestas entre v_1 e os vértices de $Q_1 \cap Q_3$ e entre v_3 com vértices de $Q_1 \cap Q_3$ são todas tomadas separadamente em Ω, o que provoca um acréscimo de $2b$ cliques em Ω, ou seja, o caso mínimo é feito supondo a existência de C.

Além disso podemos supor que $a, b, c > 0$, pois caso contrário voltaríamos a algum dos casos anteriores do Teorema, o que nos permite considerar as arestas v_1q_1 e v_2q_2, onde $q_1 \in Q_1 - Q_3$ e $q_3 \in Q_3 - Q_1$. Como tais arestas também não fazem parte das contagens anteriores, então $|\Omega| \geq |Q_2| + b + 2 = a+2b+c+2$.

Note que o CASO D é inviável se comparado ao CASO A, visto que $|\Omega|$ tem cardinalidade maior. Então o mínimo é dado pelos CASO A, B ou C, ou seja, quando Q_1, Q_2 ou Q_3 pertence à Ω.

□

Exemplo 5.3.1. *Considere os grafos G_a e G_b ilustrados na Figura 5.17.*

$$cp(G_a) = 2 + |Q_3 - Q_1| \cdot (|Q_1 \cap Q_3| - 1) + (|Q_1 - Q_3| - 1)(|Q_3 - Q_1| - 1) = 9$$

$$cp(G_b) = 1 + |Q_1 \cap Q_2| + |Q_2 \cap Q_3| = 11$$

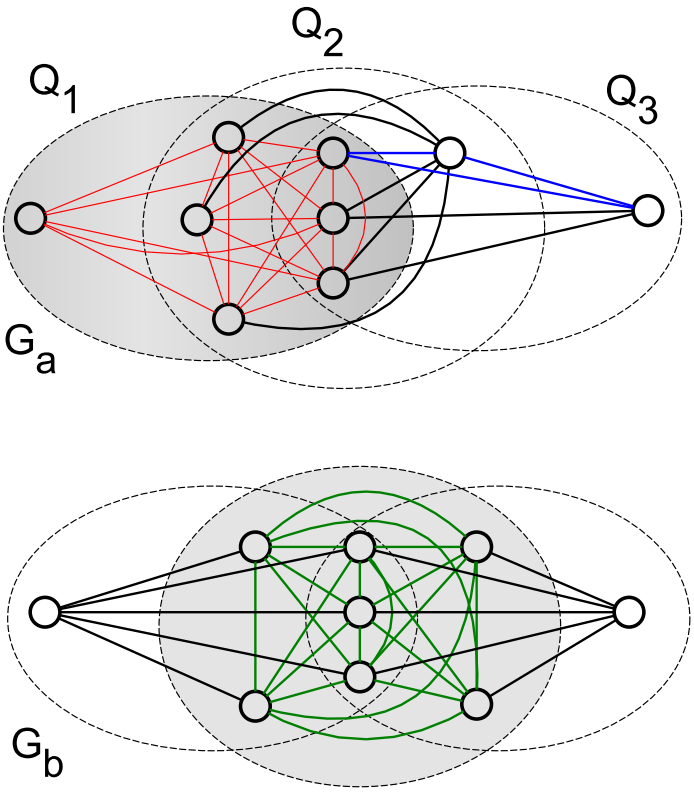

Figura 5.17: Exemplo ECP em Grafos split-indiferença

5.3.2 Subclasses do $(3, 0)$

Provamos na subseção 5.2.1 que o problema ECP é $\mathcal{NP}$-completo quando restrito à grafos $(3, 0)$. Nesta seção estudaremos a complexidade do problema em três distintas subclasses.

5.3.2.1 2-árvore

A classe das k-árvores foi introduzida por Rose em [52] e é uma generalização da classe das árvores ($k = 1$), cuja definição é dada de forma recursiva.

Definição 5.3.1. *Um grafo completo com k vértices é uma k-árvore. Uma k-árvore com $n + 1 \geq k + 1$ vértices pode ser obtida de um k-árvore G com n vértices após inserção de*

um vértice v adjacente à exatamente todos os vértices de uma k-clique Q de G, chamada joint-clique de v.

Direto da definição vemos que toda 2-árvore é 3-colorível e cordal.

Exemplo 5.3.2. *O grafo abaixo é uma 2-árvore.*

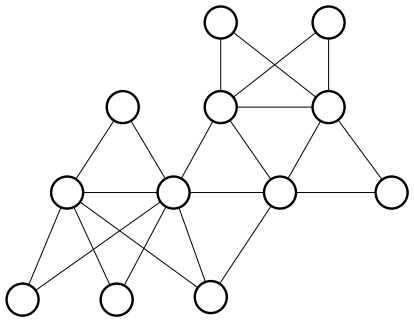

Figura 5.18: Grafo 2-árvore.

Um grafo é dito uma k-árvore parcial se é um subgrafo (não necessariamente induzido) de uma k-árvore.

Também vale lembrar a definição de grafo bloco, ou seja, aquele que toda componente biconexa é uma clique. Vale lembrar que uma componente é k-conexa quando são necessários ao menos k vértices para tornar o (sub)grafo desconexo. É fácil concluir que os grafos blocos formam uma subclasse dos grafos cordais. Observe o exemplo a seguir:

Exemplo 5.3.3. *O grafo ilustrado abaixo é um grafo bloco.*

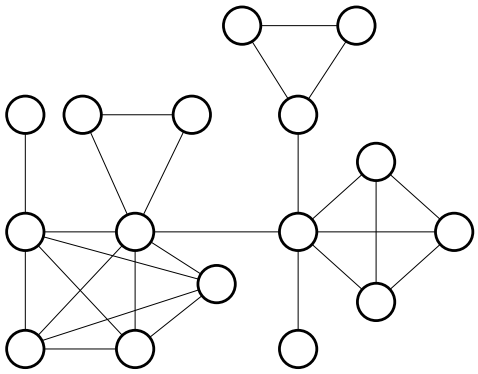

Figura 5.19: Grafo Bloco

Compreendido as definições anterior, podemos apresentar nosso próximo resultado.

Proposição 5.3.2. *O Problema ECP é polinomial quando restrito a uma* 2*-árvore parcial* G*. Além disso* $cp(G) = m - 2\alpha(\ell_3(G)))$.

Demonstração. Seja H uma 2-árvore com n vértices. Cada novo vértice z inserido em H equivale a um novo vértice v em $\ell_3(H)$, associado ao triângulo xyz em H, onde a *joint-clique* de z é a aresta $xy \in E(H)$.

Pela construção da 2-árvore é fácil ver que toda clique maximal é um triângulo, portanto para cada aresta xy existe vértice a tal que axy é um triângulo em H. Considere que o novo vértice z é inserido escolhendo a clique $\{x, y\}$ como sua *joint-clique*, conforme ilustrado na Figura 5.20.

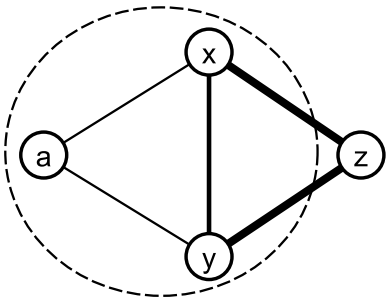

Figura 5.20: Inserção de z em uma 2-árvore

Temos dois casos a considerar:

1. O vértice z é o primeiro a escolher $\{x, y\}$ como *joint-clique*.

   Neste caso aresta xy está contida em exatamente dois triângulos $u = axy$ e $v = zxy$, o que é representado em $\ell_3(H)$ pela criação do vértice v pendente ao u.

2. O vértice z é o k-ésimo ($k \geq 2$) vértice a escolher $\{x, y\}$ como *joint-clique*.

   Digamos que $\{x, y\}$ foi escolhida por $z_1, z_2, \ldots, z_k = z$, $k \geq 2$, o que acarreta em exatamente $(k + 1)$ triângulos em H que compartilham a aresta xy. Em $\ell_3(H)$ tal operação é representada pela inserção de um novo vértice xyz_k adjacente à clique maximal formada pelos vértices referentes aos triângulos que compartilham $xy \in E(H)$. Vale lembrar que antes da primeira escolha de $\{x, y\}$, esta aresta estava contida apenas no triângulo axy, conforme ilustrado na Figura 5.21.

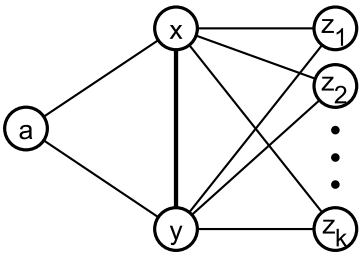

Figura 5.21: Inserção de z em uma 2-árvore cuja *joint-clique* é repetida

Logo, pela construção temos que $\ell_3(H)$ é um grafo construído pelas operações:

- $\boldsymbol{O_1}$**:** Inserção de vértice pendente;
- $\boldsymbol{O_2}$**:** Inserção de um vértice adjacente à todos os vértices de uma clique maximal.

Observe que a única forma de construir uma componente biconexa em $\ell_3(H)$ é pela operação O_2, que por sua vez resulta em clique, isto é, $\ell_3(H)$ é um grafo bloco.

Como H é K_4-livre então pelo Lema 5.2.1 e Corolário 5.1.2 temos que o Problema de ECP em G equivale ao problema MTED, que por sua vez é equivalente ao problema MIS em $\ell_3(H)$. Em [22] é provado que este último problema é polinomial quando restrito à classe de grafos cordais, que contém os grafos bloco. Logo ECP é polinomial em 2-árvore.

Seja G subgrafo de H, isto é, G uma 2-árvore parcial. Observe que $\ell_3(G)$ é subgrafo induzido de $\ell_3(H)$, pois a remoção de qualquer aresta ou vértice em H implica na remoção de um vértice em $\ell_3(H)$, tornando $\ell_3(G)$ também grafo bloco. Logo ECP é polinomial em 2-árvore parcial G e além disso $cp(G) = |E(G)| - \alpha(\ell_3(G))$. □

Em 1998, Bodlaender [4] listou várias equivalências para grafos k-árvore parcial, dentre estas vale ressaltar que esta é equivalente aos grafos com *treewidth* no máximo k. Logo a Proposição obtida acima garante que ECP é polinomial para grafos com *treewidth* no máximo 2.

5.3.2.2 Grade Triangular

O grafo infinito T^∞ é o grafo cujos vértices são representados por pares de inteiros (x, y) associados ao ponto $x \cdot \overrightarrow{v} + y \cdot \overrightarrow{u}$ do $\mathbb{R}^2$, onde $\overrightarrow{v} = (1, 0) \in \mathbb{R}^2$ e $\overrightarrow{u} = (1/2, \sqrt{3}/2) \in \mathbb{R}^2$. Dois vértices deste grafo são adjacentes se a distância euclidiana é exatamente 1, formando uma malha infinita de triângulos equiláteros, como pode ser observado na Figura 5.22.

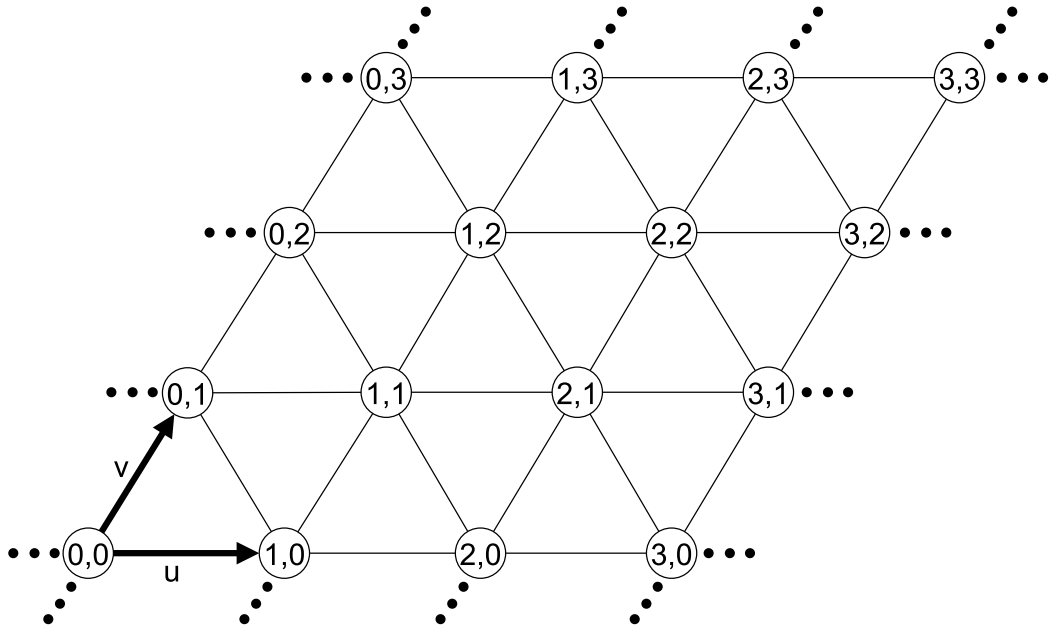

Figura 5.22: Grafo T^{∞}

Um Grafo Grade Triangular é um subgrafo induzido finito de T^{∞}. É fácil ver que todo grafo desta classe é 3-colorível, portanto forma uma subclasse do $(3, 0)$.

Exemplo 5.3.4. *O grafo abaixo é um Grafo Grade Triangular com sua* 3*-coloração.*

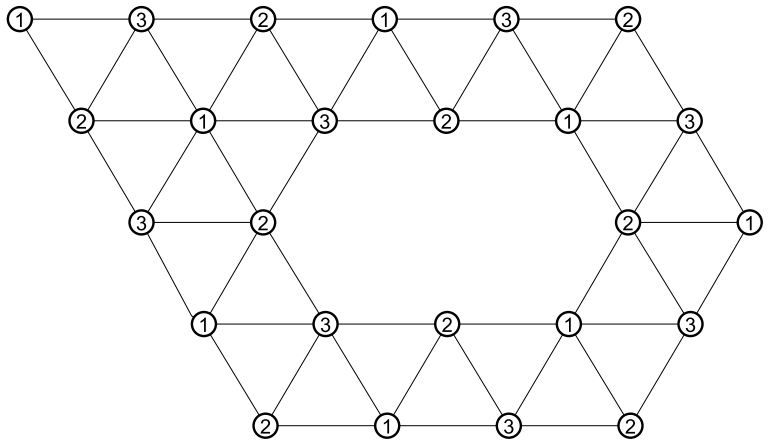

Figura 5.23: Exemplo Grafo Grade Triangular

O resultado a seguir garante que o Problema ECP é polinomial nesta classe, cuja prova é baseada em estudar o grafo triangular linha (3-linha) de um grafo Grade Triangular. Na Figura 5.24 vemos a aplicação deste operador no grafo do Exemplo 5.3.4.

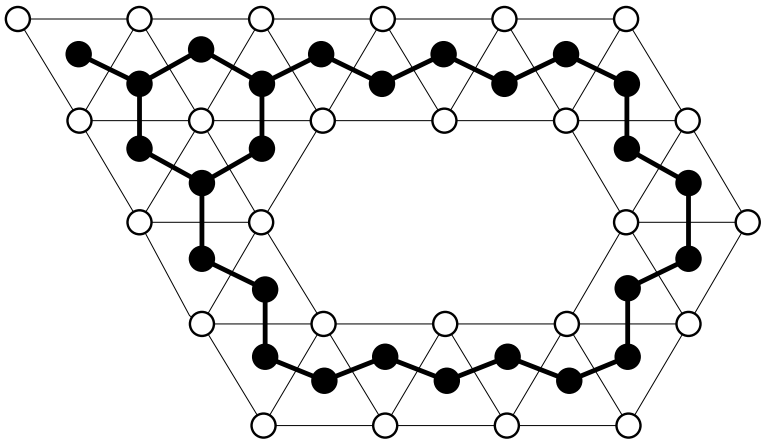

Figura 5.24: Exemplo de 3-linha aplicado em Grade Triangular

Proposição 5.3.3. *O Problema ECP é polinomial quando restrito à classe dos grafos Grade Triangular.*

Demonstração. Seja G um grafo Grade Triangular, como G é K_4-livre, então o Problema ECP em G é equivalente ao Problema MIS em $\ell_3(G)$, pelo Corolário 5.1.2. Portanto, semelhante à prova da Proposição 5.3.2, provaremos que o problema MIS é polinomial em $\ell_3(G)$.

Considere o grafo T^∞ e tome os subgrafos induzidos em vértices

$$T_i^\infty = T^\infty[\mathbb{Z} \times \{i, i+1\}], \text{ para cada } i \in \mathbb{Z},$$

que podem ser vistos como "faixas horizontais de triângulos". Observe que $T^\infty = \bigcup_{i\in\mathbb{Z}} T_i^\infty$, fazendo a identificação dos vértices de interseção. Para cada $i \in \mathbb{Z}$, vemos que o grafo triangular $\ell_3(T_i^\infty)$ é isomorfo a um caminho infinito $P_\infty = (v_1, v_2, \ldots)$, que é 2-colorível. Ao construirmos o grafo triangular de duas faixas consecutivas temos o grafo $\ell_3(T_i^\infty \cup T_{i+1}^\infty)$ obtido por dois caminhos:

$$P_\infty = v_1, v_2, \ldots \quad \text{e} \quad P'_\infty = u_1, u_2, \ldots$$

Onde as arestas entre P_∞ e P'_∞ ocorrem conforme um emparelhamento alternado, ou seja:

$$E(\ell_3(T_i^\infty \cup T_{i+1}^\infty)) = \{v_i v_{i+1}, u_i u_{i+1}; i \in \mathbb{Z}\} \cup \{v_i u_{i-1}; i \text{ é inteiro par}\}$$

O que garante que o grafo ainda é 2-colorível, pois basta fazer a alternância de cores entre caminhos consecutivos, conforme a Figura 5.25.

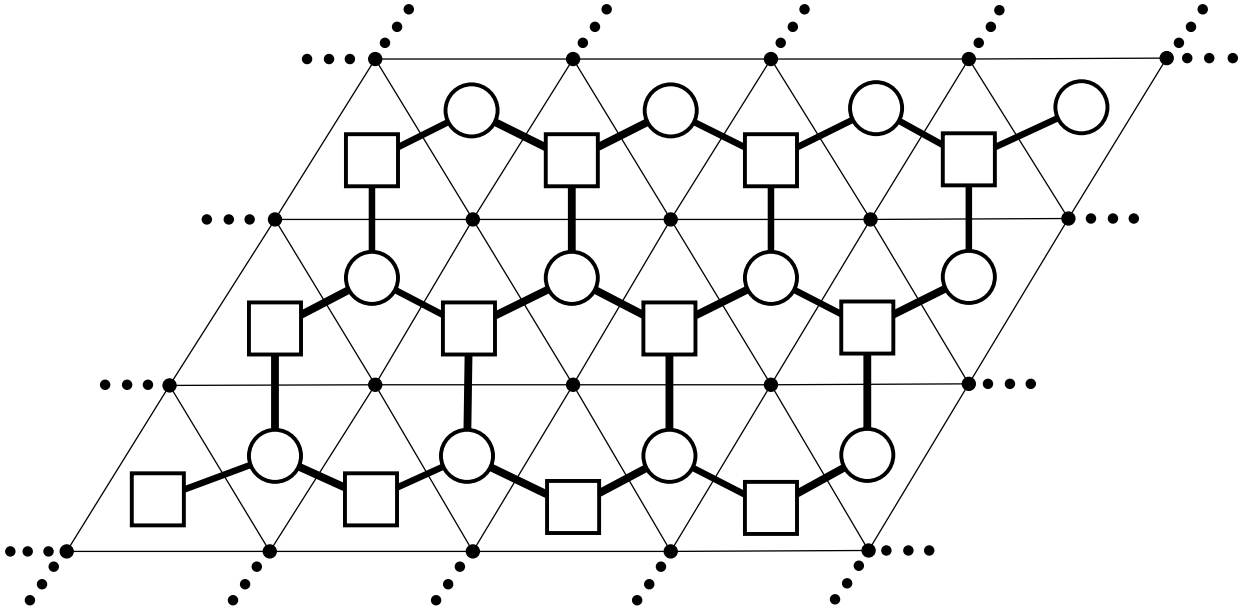

Figura 5.25: Caso base da Proposição 5.3.3

Logo o grafo infinito $\ell_3(T^\infty)$ é 2-colorível e claramente ao considerarmos qualquer grafo grade triangular G (subgrafo induzido finito), seu grafo triangular linha $\ell_3(G)$ também será 2-colorível. É fácil ver que o Problema MIS é polinomial nesta classe, basta determinar o maior conjunto independente dentre os dois únicos maximais possíveis. Donde segue o resultado desejado.

□

5.3.2.3 Grafo $\{F_2, F_4\}$-livres $(3, 0)$

Inicialmente introduziremos o conceito de grafo Gallai e Anti-gallai, cuja construção foi feita em 1967, em [20].

Definição 5.3.2. *Dado um grafo G, seu grafo Gallai $\Gamma(G)$ tem como conjunto de vértices as arestas de G, onde dois vértices são adjacentes em $\Gamma(G)$ se as respectivas arestas são adjacentes em G, mas não estão em um mesmo triângulo. O grafo anti-Gallai $\Delta(G)$ também possui como vértices as arestas de G, porém estes são adjacentes se as respectivas arestas pertencem a um triângulo de G.*

Podemos observar que $\Delta(G)$ e $\Gamma(G)$ são subgrafos de $\ell(G)$, onde $\Delta(G) \cup \Gamma(G) = \ell(G)$, considerando que vértices comuns são identificados. Esses conceitos são estudados em diversos artigos, dentre estes [36], [54] e [38].

Exemplo 5.3.5. *Na figura abaixo ilustramos um grafo G, $\ell(G)$, $\Delta(G)$ e $\Gamma(G)$.*

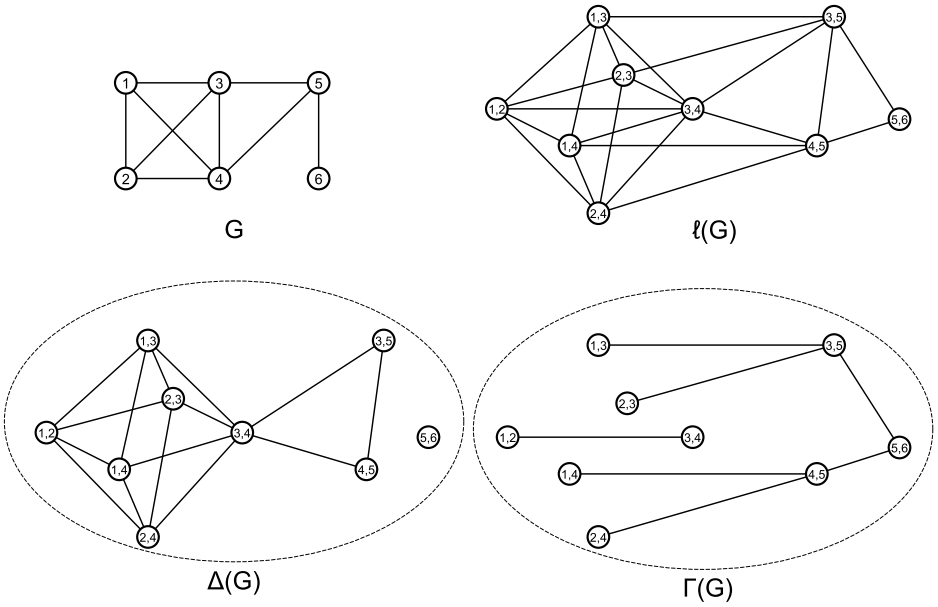

Figura 5.26: Exemplo grafo linha, Gallai e anti-Gallai de G

Lakshmanan *et. al.* em [36] exibem uma lista de 5 grafos proibidos em G de forma que seu anti-Gallai seja cografo, conforme enunciado abaixo.

Teorema 5.3.2. *[36] Seja G um grafo. O anti-gallai $\Delta(G)$ é um cografo se, e somente se, G não induz os subgrafos F_i, $i = 1, 2, 3, 4, 5$, ilustrados na Figura 5.27*

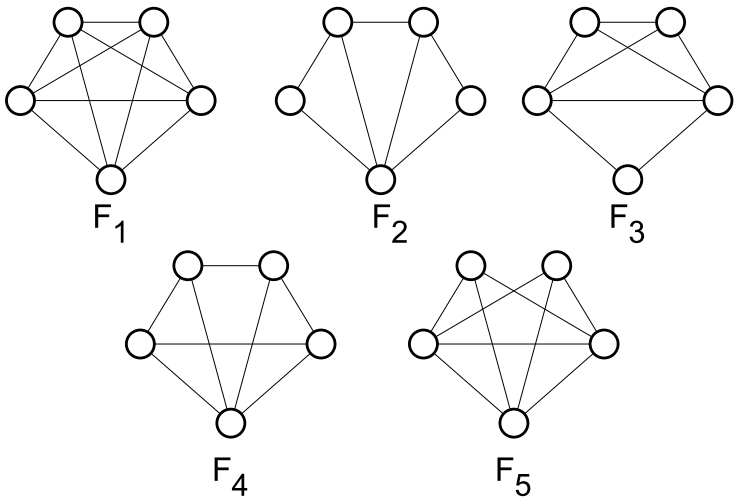

Figura 5.27: Grafos F_i, $i = 1, 2, 3, 4, 5$ proibidos do Teorema 5.3.2

O autor Van Bang Le prova em [38] o seguinte resultado.

Proposição 5.3.4. *[38] Para todo grafo G com número clique $\omega(G) \geq 1$ vale:*

$$\omega(\Delta(G)) = \begin{cases} \omega(G) - 1, \text{ se } \omega(G) \neq 3 \\ 3 \text{ , se } \omega(G) = 3 \end{cases} . \tag{5.2}$$

No início deste capítulo comentamos sobre o número mínimo de cliques necessárias para particionar os vértices em cliques, descrito abaixo:

Problema 5.3.1. *(**VCP** - vertice clique partition)*
INSTÂNCIA: Grafo G e inteiro $k \geq 0$.
QUESTÃO: O número mínimo de cliques que particiona os vértices de G é menor ou igual à k?

Mostraremos a seguir a relação entre os Problemas ECP e VCP para grafos K_4-livre.

Proposição 5.3.5. *O problema ECP em G K_4-livre é equivalente ao problema VCP no anti-Gallai $\Delta(G)$.*

Demonstração. Seja G um grafo livre de K_4, então pela Proposição 5.3.4 garantimos que $\Delta(G)$ também é livre de K_4, pois $\omega(G) \leq 3$ garante que $\omega(\Delta(G)) \leq 3$.

A equivalência dos problemas é trivial no caso em que G induz até um triângulo.

Vamos mostrar que dois triângulos em G disjuntos em arestas é equivalente à dois triângulos em $\Delta(G)$ disjuntos em vértices. Considere o caso em que G induz dois triângulos distintos induzidos em vértices $T_1 = a_1, b_1, c_1$ e $T_2 = a_2, b_2, c_2$.

Por definição, em $\Delta(G)$ temos os vértices $T_1^\Delta = \{a_1b_1, a_1c_1, b_1c_1\}$ e $T_1^\Delta = \{a_2b_2, a_2c_2, b_2c_2\}$ que também formam triângulos.

Se T_1 e T_2 possuem uma aresta em comum, digamos $a_1 = a_2$ e $b_1 = b_2$, então os T_1^Δ e T_2^Δ em $\Delta(G)$ compartilham um vértice comum $a_1b_1 = a_2b_2$. Reciprocamente se T_1^Δ e T_2^Δ possuem um vértice comum em $\Delta(G)$, digamos $a_1b_1 = a_2b_2$ então existe uma aresta em G que pertence a dois triângulos, ou seja, $a_1 = a_2$ e $b_1 = b_2$.

Logo dois triângulos em G disjuntos em arestas é equivalente à dois triângulos em $\Delta(G)$ disjuntos em vértices. O que garante a equivalência desejada. □

Com o auxílio dos resultados acima provamos a complexidade do problema ECP em uma subclasse de grafos $(3, 0)$.

Proposição 5.3.6. *O problema ECP é polinomial em grafos* 3*-colorível* $\{F_2, F_4\}$*-livres (veja Figura 5.27).*

Demonstração. Seja G 3-colorível $\{F_2, F_4\}$-livres, então pelo Teorema 5.3.2 temos $\Delta(G)$ cografo.

E bem conhecido na literatura que o problema de coloração de vértices é polinomial em cografos, portanto VCP também é nesta classe. Pela Proposição 5.3.5 vemos que este problema é equivalente à ECP para G 3-colorível, em particular. Logo o problema de ECP é polinomial para classe desejada. □

5.3.2.4 Relação entre subclasses de $(3, 0)$

Nesta subseção provamos que o Problema ECP se torna polinomial quando restrito a algumas subclasses do $(3, 0)$, são estas: 2-árvore, Grade Triangular e $\{F_2, F_4\}$-livres$\cap(3, 0)$. O estudo destas subclasses foi motivado por buscas na página *"Information System on Graph Classes and their Inclusions"* [15]. O Diagrama da Figura 5.28 mostra exemplo de grafos que pertencem a exatamente uma das três subclasses estudadas.

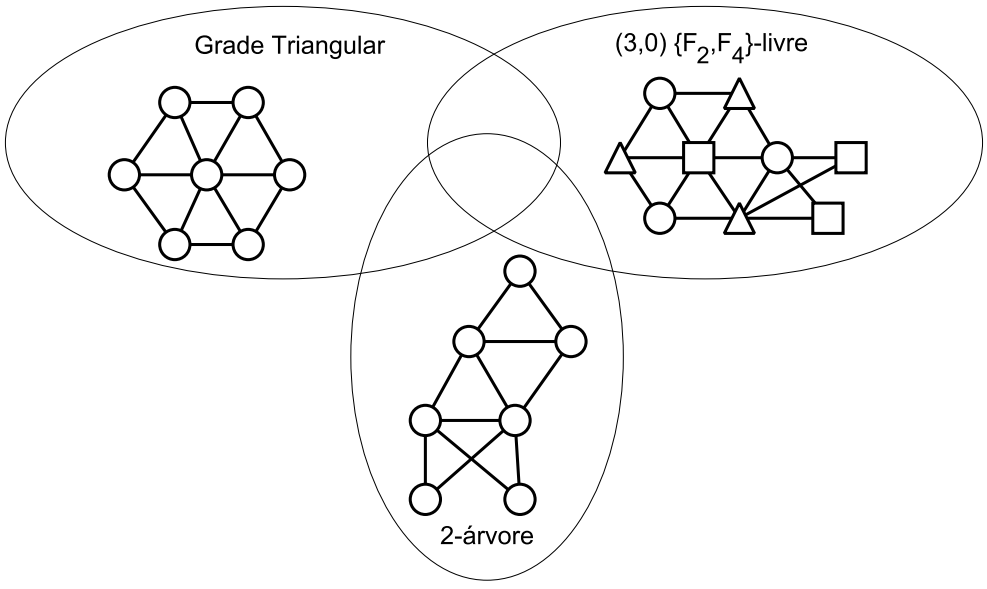

Figura 5.28: Diagrama para subclasses do $(3, 0)$ com ECP polinomial

Capítulo 6

Conclusão

Neste trabalho apresentamos dois principais resultados.

O primeiro foi desenvolvido no capítulo 3, com o algoritmo 3.10 capaz de gerar todos os cografos com n vértices não isomorfos, garantido pelo Teorema 3.4.1 e com tempo linear em n para geração de dois cografos consecutivos, pela Proposição 3.4.1. Para elaboração do algoritmo introduzimos novos conceitos como *ordenação de vértices em árvore* e *ordenação de cografos*, *pivô de uma árvore*.

No capítulo 4 implementamos e aplicamos o algoritmo de geração de cografos em um problema envolvendo número b-cromático e raio espectral em cografos. Constatamos que todos os cografos conexos até 21 vértices satisfazem a desigualdade $\chi_b(G) \leq \lceil \lambda(G) \rceil + 1$, o que permitiu a formulação de uma conjectura. Além disso vimos que é insignificante a quantidade de cografos que não satisfazem a desigualdade $m(G) \leq \lceil \lambda(G) \rceil + 1$, o que sugere uma pesquisa na estrutura desses cografos.

Outro importante resultado foi o Teorema 5.2.1 junto à teoria desenvolvida no capítulo 5, mais especificamente na Seção 5.2, onde estudamos a complexidade do problema de partição de arestas em cliques para grafos (k, ℓ) para cada $k \geq 0$ e $\ell \geq 0$, explicitando quando o problema deixa de ser polinomial e passa a ser $\mathcal{NP}$-completo . Nossos resultados estendem a complexidade obtida em [56], onde é provado que o problema é $\mathcal{NP}$-completo para grafos *split* (isto é, $(1, 1)$). O Teorema 5.2.1 que garante complexidade $\mathcal{NP}$-completo para grafos 3-colorível (ou $(3, 0)$), cuja importância é pelo fato de provarmos que o problema ECP permanece $\mathcal{NP}$-completo ainda quando tomamos uma subclasse dos grafos livres de K_4, cuja complexidade para o problema foi provada em 1988 em [41]. Ainda neste capítulo obtemos diversas subclasses do $(3, 0)$ e uma subclasse do $(1, 1)$ onde o problema deixa de ser $\mathcal{NP}$-completo e se torna polinomial.

Como trabalho futuro propomos a tentativa de demonstração da conjectura 4.4.1 e a caracterização dos " poucos " cografos conexos que não satisfazem a desigualdade 4.2. Quanto à segunda parte do trabalho propomos o estudo da complexidade de ECP na classe dos cografos.

Referências

[1] Alkhateeb, M.; Kohl, A. Upper bounds on the b-chromatic number and results for restricted graph classes. *Discussiones Mathematicae Graph Theory 31*, 4 (2011), 709–735.

[2] Anand, P.; Escuadro, H.; Gera, R.; Hartke, S. G.; Stolee, D. On the hardness of recognizing triangular line graphs. *Discrete Mathematics 312*, 17 (2012), 2627–2638.

[3] Bagga, J. Old and new generalizations of line graphs. *International Journal of Mathematics and Mathematical Sciences 2004*, 29 (2004), 1509–1521.

[4] Bodlaender, H. L. A partial k-arboretum of graphs with bounded treewidth. *Theoretical computer science 209*, 1-2 (1998), 1–45.

[5] Bondy, J. A.; Murty, U. S. R. *Graph theory with applications*, vol. 290. Macmillan London, 1976.

[6] Bonomo, F.; Duran, G.; Maffray, F.; Marenco, J.; Valencia-Pabon, M. On the b-coloring of cographs and p 4-sparse graphs. *Graphs and Combinatorics 25*, 2 (2009), 153–167.

[7] Brandstädt, A.; Szymczak, T., et al. The complexity of some problems related to graph 3-colorability. *Discrete Applied Mathematics 89*, 1-3 (1998), 59–73.

[8] Bravo, R. S. F.; Klein, S.; Nogueira, L. T.; Protti, F. Characterization and recognition of p4-sparse graphs partitionable into k independent sets and l cliques. *Discrete Applied Mathematics 159*, 4 (2011), 165–173.

[9] Brouwer, A. E.; Haemers, W. H. *Spectra of graphs*. Springer Science & Business Media, 2011.

[10] Cioaba, S. M. *The NP-completeness of some edge-partitioning problems*. Tese de Doutorado, M. Sc. Thesis, Queen's University at Kingston, Canada, 2002.

[11] Corneil, D.; Perl, Y.; Stewart, L. Cographs: recognition, applications and algorithms. *Congressus Numer 43* (1984), 249–258.

[12] Corneil, D. G.; Lerchs, H.; Burlingham, L. S. Complement reducible graphs. *Discrete Applied Mathematics 3*, 3 (1981), 163–174.

[13] Cvetković, D.; Rowlinson, P. The largest eigenvalue of a graph: A survey. *Linear and multilinear algebra 28*, 1-2 (1990), 3–33.

[14] Dantas, S.; de Figueiredo, C. M.; Faria, L. On decision and optimization (k, l)-graph sandwich problems. *Discrete Applied Mathematics 143*, 1-3 (2004), 155–165.

[15] de Ridder, H. N., et al. Information System on Graph Classes and their Inclusions (ISGCI). `http://www.graphclasses.org`.

[16] de Souza Francisco, R.; Klein, S.; Nogueira, L. T. Characterizing (k, l)–partitionable cographs. *Electronic Notes in Discrete Mathematics 22* (2005), 277–280.

[17] Erdos, P.; Goodman, A. W.; Pósa, L. The representation of a graph by set intersections. *Canad. J. Math 18*, 106-112 (1966), 86.

[18] Figueroa, A.; Borneman, J.; Jiang, T. Clustering binary fingerprint vectors with missing values for dna array data analysis. *Journal of Computational biology 11*, 5 (2004), 887–901.

[19] Fleischer, R.; Wu, X. Edge clique partition of k 4-free and planar graphs. In *Computational Geometry, Graphs and Applications*. Springer, 2011, pp. 84–95.

[20] Gallai, T. Transitiv orientierbare graphen. *Acta Mathematica Academiae Scientiarum Hungarica 18*, 1-2 (1967), 25–66.

[21] Garey, M. R. A guide to the theory of np-completeness. *Computers and intractability* (1979).

[22] Gavril, F. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM Journal on Computing 1*, 2 (1972), 180–187.

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

[24] Hoàng, C. T.; Kouider, M. On the b-dominating coloring of graphs. *Discrete Applied Mathematics 152*, 1 (2005), 176–186.

[25] Hoover, D. N. Complexity of graph covering problems for graphs of low degree. *Journal of Combinatorial Mathematics and Combinatorial Computing 11*, 187-208 (1992), 7.

[26] Horn, R. A.; Horn, R. A.; Johnson, C. R. *Matrix analysis*, vol. 1. Cambridge university press, 1990.

[27] Irving, R. W.; Manlove, D. F. The b-chromatic number of a graph. *Discrete Applied Mathematics 91*, 1 (1999), 127–141.

[28] Jacobs, D. P.; Trevisan, V.; Tura, F. C. Eigenvalue location in cographs. *Discrete Applied Mathematics* (2017).

[29] Jamison, B.; Olariu, S. A tree representation for p4-sparse graphs. *Discrete Applied Mathematics 35*, 2 (1992), 115–129.

[30] Jones, Á. A.; Del-Vecchio, R. R. Propriedades espectrais dos grafos p4-esparsos. *Dissertação de Mestrado em Matemática, Universidade Federal Fluminense* (2014).

[31] Jones, Á. A.; Protti, F.; Del-Vecchio, R. R. Uma relação entre b-coloração e teoria espectral em cografos. In *XLIX Simpósio Brasileiro de Pesquisa Operacional* (2017), pp. 3383–3394.

[32] Jones, Á. A.; Protti, F.; Del-Vecchio, R. R. Cograph generation with linear delay. *Theoretical Computer Science* (2018).

[33] Karp, R. M. Reducibility among combinatorial problems. In *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[34] Lakshmanan, A.; Bujtás, C.; Tuza, Z. Generalized line graphs: Cartesian products and complexity of recognition. *The Electronic Journal of Combinatorics 22*, 3 (2015), 3–33.

[35] Lakshmanan, A.; Bujtás, C.; Tuza, Z. Induced cycles in triangle graphs. *Discrete Applied Mathematics 209* (2016), 264–275.

[36] Lakshmanan, A.; Rao, S.; Vijayakumar, A. Gallai and anti-gallai graphs of a graph. *Math. Bohem. 132*, 1 (2007), 43–54.

[37] Lê, V. B. Perfect k-line graphs and k-total graphs. *Journal of graph theory 17*, 1 (1993), 65–73.

[38] Le, V. B. Gallai graphs and anti-gallai graphs. *Discrete Mathematics 159*, 1-3 (1996), 179–189.

[39] Lerchs, H. On cliques and kernels. *Department of Computer Science, University of Toronto* (1971).

[40] Lovász, L. On covering of graphs. In *Theory of Graphs (Proc. Colloq., Tihany, 1966)* (1968), Academic Press New York, pp. 231–236.

[41] Ma, S.; Wallis, W.; Wu, J.-L. The complexity of the clique partition number problem. *Congressus Numer 67* (1988), 59–66.

[42] Markenzon, L. *Ferramentas Estruturais em Grafos Cordais*, vol. 82. Notas em Matemática Aplicada - SBMAC, 2016. 96 p.

[43] McKay, B. graph6 and sparse6 graph formats. *Computer Science Department, Australian National University, September 12* (2007), 2007. http://users.cecs.anu.edu.au/ bdm/data/formats.html.

[44] Merris, R. Laplacian graph eigenvectors. *Linear algebra and its applications 278*, 1 (1998), 221–236.

[45] Monson, S. D.; Pullman, N.; Rees, R. A survey of clique and biclique coverings and factorizations of (0, 1)-matrices. *Bull. Inst. Combin. Appl 14* (1995), 17–86.

[46] Orlin, J. Contentment in graph theory: covering graphs with cliques. In *Indagationes Mathematicae (Proceedings)* (1977), vol. 80(5), Elsevier, pp. 406–424.

[47] Ortiz, Z. C.; Maculan, N.; Szwarcfiter, J. L. Characterizing and edge-colouring split-indifference graphs. *Discrete applied mathematics 82*, 1-3 (1998), 209–217.

[48] Pullman, N. J. Clique coverings of graphs—a survey. In *Combinatorial Mathematics X*. Springer, 1983, pp. 72–85.

[49] Pullman, N. J. Clique covering of graphs iv. algorithms. *SIAM Journal on Computing 13*, 1 (1984), 57–75.

[50] Ravelomanana, V.; Thimonier, L. Asymptotic enumeration of cographs. *Electronic Notes in Discrete Mathematics 7* (2001), 58–61.

[51] Roberts, F. S. Applications of edge coverings by cliques. *Discrete applied mathematics 10*, 1 (1985), 93–109.

[52] Rose, D. J. On simple characterizations of k-trees. *Discrete mathematics 7*, 3-4 (1974), 317–322.

[53] S. H. Ma, W. D. W. Clique numbers of threshold graphs. *Caribbean J. Math. 5(1)* (1986), 29–45.

[54] Sun, L. Two classes of perfect graphs. *Journal of Combinatorial Theory, Series B 53*, 2 (1991), 273–292.

[55] Uiyyasathain, C. *Maximal-clique partitions.* Tese de Doutorado, University of Colorado at Denver, 2003.

[56] Wallis, W.; Wu, J.-L. On clique partitions of split graphs. *Discrete mathematics 92*, 1-3 (1991), 427–429.

[57] Wilf, H. S. The eigenvalues of a graph and its chromatic number. *J. London Math. Soc 42*, 1967 (1967), 330.