Tratabilidade parametrizada de problemas NP-completos

Maise Dantas da Silva

Mestrado em Informática
NCE/IM - UFRJ

Orientadores:
Fábio Protti, D.Sc.
Lilian Markenzon, D.Sc.

Rio de Janeiro, RJ - Brasil
2004

Tratabilidade parametrizada de problemas NP-completos

Maise Dantas da Silva

Dissertação submetida ao Corpo Docente do Núcleo de Computação Eletrônica – Instituto de Matemática da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários para obtenção do grau de Mestre em Informática.

Aprovada por:

Prof. Fábio Protti, D.Sc. - Orientador

Prof[a]. Lilian Markenzon. D.Sc. - Orientadora

Prof. Carlos Alberto de Jesus Martinhon, D.Sc.

Prof[a]. Nair Maria Maia de Abreu, D.Sc.

Rio de Janeiro, RJ - Brasil
2004

Silva, Maise Dantas da
Tratabilidade parametrizada de problemas NP-completos / Maise Dantas da Silva.
Rio de Janeiro: UFRJ IM, 2004
Dissertação – Universidade Federal do Rio de Janeiro, IM / NCE.
1. Introdução
2. Complexidade parametrizada
3. Modificação de grafos por propriedade hereditária
4. O problema de cordalização(k)
5. Algoritmos por parâmetro fixo para obtenção de grafos aglomerados
6. Considerações finais
(Mestrado - UFRJ/IM/NCE) I. Título.

A minha família e ao Samuel.

Agradecimentos

A Deus, por possibilitar que meu sonho se tornasse realidade.

A minha família, pelo apoio às minhas escolhas, por estar ao meu lado a cada dia.

Ao Samuel, pela simples presença na minha vida; pelo amor, apoio e cumplicidade.

Aos meus orientadores, Fábio Protti e Lilian Markenzon, pelo acolhimento e incentivo, pela paciência e dedicação a mim e a este projeto.

Aos professores e funcionários do NCE, pela presteza e atenção dispensadas aos alunos.

Aos amigos que fiz durante o mestrado, especialmente ao Marcelo, Juliana e Vinícius. A união nas horas de dificuldade nos fizeram mais fortes, tornando mais fácil superar os desafios.

Resumo

Este trabalho visa apresentar uma nova forma de abordar problemas NP-completos, através de suas versões parametrizadas. A complexidade parametrizada, proposta por Downey e Fellows, oferece a possibilidade de desenvolvimento de algoritmos polinomiais para muitos destes problemas, quando considerados juntamente com um parâmetro. Além da teoria de complexidade parametrizada, alguns exemplos de problemas NP-completos que se tornam tratáveis quando parametrizados são abordados. Como primeiro exemplo, demonstramos a tratabilidade de problemas parametrizados de modificação tais que, a partir de um grafo de entrada e um parâmetro k, deseja-se obter, através de no máximo k alterações, um grafo resultante que pertença a uma família de grafos que satisfazem uma propriedade hereditária Π caracterizada por um conjunto proibido finito. Em seguida, uma prova da tratabilidade por parâmetro fixo para o problema de cordalização, realizada por H. Kaplan, R. Shamir e R. E. Tarjan, é estudada. Devido a sua importância, alguns resultados conhecidos sobre a versão clássica deste problema, que é NP-completo, também são revistos. Por último, são estudados algoritmos tratáveis para o problema parametrizado de obtenção de grafos aglomerados, recentemente propostos por J. Gramm, J. Guo, F. Hüffner e R. Niedermeier.

Palavras-chave: tratabilidade por parâmetro fixo, cordalização, grafos aglomerados.

Abstract

The objective of this dissertation is to present a new form of dealing with NP-complete problems, considering its parameterized versions. The parameterized complexity, introduced by Downey and Fellows, offers the possibility of developing polynomial time algorithms for many of these problems, when they are considered with a parameter. Besides the parameterized complexity theory, some examples of NP-complete problems that become tratable when they are parameterized are shown. As a first example, we considered the tractability of parameterized modification problems such that, starting from an input graph and a parameter k, the goal is to obtain, by at most k modifications, a resulting graph that belongs to a family of graphs satisfying a hereditary property Π characterized by a finite forbidden set. In the sequence, a proof of the fixed parameter tractability of the fill-in problem, achieved by H. Kaplan, R. Shamir and R. E. Tarjan, is studied. Due to its importance, some well-known results about the classic version of this problem, which is NP-complete, are also reviewed. Last, tractable fixed-parameter algorithms for clique generation, recently proposed by J. Gramm, J. Guo, F. Hüffner and R. Niedermeier, are presented.

Keywords: fixed-parameter tractability, fill-in, cluster graph.

Sumário

Lista de Figuras

Capítulo 1

Introdução

Nesta dissertação, será estudada a teoria de complexidade parametrizada, proposta por Downey e Fellows, além de alguns problemas parametrizados tratáveis, cuja versão clássica é NP-completa. Devido a sua importância em várias áreas de aplicação, é dada uma atenção especial à versão clássica do problema de cordalização, pela apresentação de alguns de seus resultados. Assim, este trabalho está organizado da seguinte forma:

Neste capítulo, são introduzidos alguns conceitos básicos sobre a complexidade clássica de algoritmos, a fim de possibilitar a compreensão da importância prática de se obter algoritmos tratáveis por parâmetro fixo para problemas NP-completos. São fornecidas também as definições e notações sobre grafos utilizadas ao longo deste trabalho. As definições específicas a cada problema parametrizado abordado são vistas nos capítulos correspondentes.

O Capítulo 2 traz as definições de problemas parametrizados e das classes em que eles são organizados. Para a classe FPT (Fixed Parameter Tractability), que agrupa os problemas tratáveis de acordo com a complexidade parametrizada, são vistas duas técnicas de prova, amplamente usadas em vários problemas parametrizados tratáveis, incluindo os abordados nos próximos capítulos.

O Capítulo 3 apresenta a prova de tratabilidade, feita por L. Cai [2], do problema de modificação $\Pi(i,j,k)-$grafo, sendo Π uma propriedade hereditária que admite uma caracterização por conjunto proibido finito, e os parâmetros i,j,k, inteiros não negativos. Neste problema, i representa o

número máximo de vértices que podem ser retirados do grafo G de entrada; j e k representam o máximo de arestas que podem ser retiradas e acrescentadas a G, respectivamente. No mesmo artigo, Cai desenvolveu também uma prova de tratabilidade do problema parametrizado de cordalização(k). No entanto, optamos por apresentar (no capítulo seguinte) a prova elaborada por H. Kaplan, R. Shamir e R. E. Tarjan, em [12], uma vez que esta solução serviu de base para o desenvolvimento do primeiro algoritmo aproximativo polinomial para este problema, elaborado por A. Natanzon, R Shamir e R. Sharan em 2000 [13].

Nos Capítulos 4 e 5, são abordados dois exemplos de aplicação das técnicas de prova de tratabilidade, vistas no Capítulo 2.

O Capítulo 4 é iniciado com a apresentação de alguns resultados relativos a grafos cordais e ao problema de cordalização, obtidos ao longo dos mais de 30 anos em que este problema tem sido intensamente estudado. Alguns algoritmos também são apresentados. Em seguida, apresentamos um estudo detalhado da prova de tratabilidade citada acima, publicada em 1999 [12].

O Capítulo 5 apresenta o problema de obtenção de grafos aglomerados, que consiste na transformação de um dado grafo de entrada em uma união de cliques disjuntas. São vistos dois algoritmos para duas versões parametrizadas semelhantes deste problema, desenvolvidos por J. Gramm, J. Guo, F. Hüffner e R. Niedermeier [11], em um trabalho recentemente aceito para publicação.

1.1 Complexidade de algoritmos

Seja um problema $\Pi(D, Q)$, onde D é o conjunto de dados e Q é a questão do problema. Os dados específicos que constituem uma entrada para Π formam uma *instância* do problema. Dizemos que Π é um *problema de decisão* se seu objetivo for responder *sim* ou *não* à questão Q. Se for possível desenvolver um algoritmo com tempo de execução polinomial no tamanho da entrada para resolver este problema, então Π pertence a uma classe denominada P. Se Π admitir um algoritmo que cheque uma solução em tempo polinomial no tamanho da entrada, dizemos que Π pertence à classe NP. É

importante observarmos que o fato de $\Pi \in NP$ não implica que uma resposta para Π possa ser encontrada rapidamente. Assim, temos que $P \subseteq NP$, mas não se sabe se $NP \subseteq P$.

Sejam $\Pi_1(D_1, Q_1)$ e $\Pi_2(D_2, Q_2)$ dois problemas de decisão. Uma *transformação polinomial* de Π_1 em Π_2 é uma função $f : D_1 \to D_2$, computada em tempo polinomial, tal que qualquer instância $I \in D_1$ produz resposta *sim* para Π_1 se e somente se $f(I)$ também produzir resposta *sim* para Π_2.

Dado um problema de decisão Π, temos que Π é *NP-completo* se $\Pi \in NP$ e, para todo problema de decisão $\Pi' \in NP$, existir uma transformação polinomial de Π' para Π.

1.2 Definições básicas e notações

Um *grafo* é um par $G = (V, E)$, onde V é um conjunto finito de $n = |V|$ elementos chamados *vértices* e

$$E \subseteq \{(v, w) | v, w \in V, v \neq w\}$$

é um conjunto de $m = |E|$ pares de vértices não ordenados chamados *arestas*. Dado $v \in V$, o conjunto

$$adj(v) = \{w \in V | (v, w) \in E\}$$

é o conjunto de vértices *adjacentes* ou *vizinhos* a v (também chamado $N(v)$). O *grau* $d(v)$ de um vértice v é o seu número de vizinhos. Denotamos $N[v] = N(v) \cup \{v\}$. Se $A \subseteq V$, o *subgrafo induzido* $G[A]$ (ou G_A) de G é o subgrafo $G[A] = (A, E(A))$, onde $E(A) = \{(x, y) \in E | x, y \in A\}$. Denotamos por $V(G)$ o conjunto de vértices do grafo G e $E(G)$ o conjunto de arestas.

Para vértices distintos $v, w \in V$, um *caminho* v, w de comprimento l, chamado *l-caminho*, é uma seqüência de vértices distintos $\mu = [v = v_1, v_2, \ldots, v_{l+1} = w]$ tal que $v_i \in adj(v_{i+1})$ para $i = 1, 2, \ldots, l$. Similarmente, um *ciclo* de comprimento l (*l-ciclo*) é uma seqüência de vértices distintos $\mu = [v_1, v_2, \ldots, v_l]$ tal que $v_i \in adj(v_{i+1})$ para $i = 1, 2, \ldots, l-1$ e $v_1 \in adj(v_l)$. Uma *corda* de um ciclo em G é uma aresta entre dois vértices

não consecutivos no ciclo. Uma *clique* é um subgrafo completo H de G. Caso $|H| = k$, então a chamamos de k-*clique*.

Ao longo deste trabalho assume-se que G é um grafo *conexo*, ou seja, para cada par de vértices distintos $v, w \in V$ existe um caminho v, w.

Capítulo 2

Complexidade parametrizada

2.1 Introdução

A teoria de complexidade parametrizada foi proposta por Downey e Fellows [8], surgindo como uma alternativa promissora para se trabalhar com problemas NP-completos. O interesse em parametrizar tais problemas se deve ao fato de que, em muitos casos, somente uma pequena faixa de valores de parâmetro é realmente importante na prática. Logo, a (aparente) intratabilidade desses problemas no caso geral é indevidamente pessimista. A principal idéia da complexidade parametrizada é analisar mais profundamente a *estrutura* da entrada. Esta não é considerada simplesmente como uma entidade; um grafo, por exemplo, mas como um grafo com um parâmetro adicional. Assim, tenta-se limitar a explosão combinatória aparentemente inevitável na solução do problema. Tipicamente, os problemas seguem a seguinte forma:

Entrada: $\langle x, k \rangle$
Parâmetro: k
Questão: $\langle x, k \rangle \in L$?

A questão é tentar olhar para o problema *por camadas*. Ou seja, deseja-se saber qual o procedimento para um k *fixo*. Os três exemplos a seguir serviram de motivação original:

Cobertura de vértices parametrizado
Instância: Um grafo $G = (V, E)$.
Parâmetro: Um inteiro positivo k.
Questão: G possui uma cobertura de vértices de tamanho $\leq k$? (Uma cobertura de vértices de um grafo G é uma coleção de vértices V' de G tal que, para todas as arestas (v_1, v_2) de G, $v_1 \in V'$ e/ou $v_2 \in V'$.)

Conjunto dominante parametrizado
Instância: Um grafo G.
Parâmetro: Um inteiro positivo k.
Questão: G possui um conjunto dominante de tamanho k? (Um conjunto dominante é um conjunto $V' \subseteq V(G)$ onde, para cada $u \in V(G) - V'$, existe um $v \in V'$ tal que $(u, v) \in E(G)$.)

Conjunto independente parametrizado
Instância: Um grafo G.
Parâmetro: Um inteiro positivo k.
Questão: G possui um conjunto de k vértices $\{x_1, \ldots, x_k\}$ tal que para todo $i \neq j$, x_i não é adjacente a x_j?

A expressão *parametrizado* será omitida quando estiver claro no contexto que se trata da versão parametrizada de um problema.

As definições básicas de complexidade parametrizada são dadas a seguir.

Definição 2.1 [7] *Uma* linguagem parametrizada *é um conjunto* $L \subseteq \Sigma^* \times \Sigma^*$, *onde* Σ *é um alfabeto fixo.*

Se L é uma linguagem parametrizada e $(x, k) \in L$, então x é referente à *parte principal* e k é referente ao *parâmetro.* Por ser mais conveniente, além de não fazer diferença perante a teoria, consideraremos que o parâmetro k é um inteiro, o que equivale a definirmos uma linguagem parametrizada como um subconjunto de $\Sigma^* \times N$.

Definição 2.2 [9] *Uma linguagem parametrizada* L *é* tratável por parâmetro fixo *se for possível determinar em um tempo* $f(k)n^\alpha$ *se* $(x, k) \in L$, *onde*

$n = |x|$, α é uma constante independente de n e de k e f é uma função arbitrária. A família das linguagens parametrizadas tratáveis por parâmetro fixo é denotada por FPT.

Podemos notar que a definição de FPT permanece inalterada se substituirmos $f(k)n^\alpha$ por $f(k) + n^\alpha$.

Alguns problemas, em sua definição clássica, possuem como parte da entrada mais de um parâmetro. Ao serem parametrizados, estes problemas podem gerar diferentes versões (dependendo de qual parâmetro passa a ser considerado fixo), que pertencem a famílias diferentes da hierarquia de complexidade parametrizada. Outros problemas aparentemente não pertencem a FPT, como, por exemplo, conjunto dominante. Assim como é feito com a complexidade de tempo polinomial, podemos encontrar evidências para a *intratabilidade* por parâmetro fixo, estudando as noções apropriadas de transformações de problemas.

Definição 2.3 [9] *Uma* transformação paramétrica *(ou* redução uniforme*) de uma linguagem parametrizada L em uma linguagem parametrizada L' é um algoritmo que transforma uma entrada consistindo no par (x, k) em um par (x', k'), tal que:*
1. $(x, k) \in L$ se e somente se $(x', k') \in L'$,
2. $k' = g(k)$ é uma função somente em k, e
3. a computação da transformação é executada em um tempo $f(k)n^\alpha$, onde $n = |x|$, α é uma constante independente de n e de k, e f é uma função arbitrária.

Como exemplo, podemos citar uma transformação paramétrica bastante elaborada (segundo [9]) desenvolvida por Downey e Fellows, do problema parametrizado de clique para o problema parametrizado de conjunto dominante, mapeando (G, k) para (G', k'), onde $k' = 2k$. Já na direção oposta, dificilmente existe uma transformação paramétrica, uma vez que o primeiro problema pertence à família chamada $W[1]-$completo, enquanto o segundo pertence a $W[2]-$completo. Tais famílias serão vistas na seção 2.3.

Consideremos agora o grafo $G = (V, E)$ e uma k-clique $V' \subseteq V$. O conjunto de vértices V' é uma k-clique em G se e somente se $V - V'$ for uma cobertura de vértices no grafo complementar G', onde os vértices são adjacentes se e somente se não o forem em G. Isto nos dá uma fácil redução em tempo polinomial do problema parametrizado de clique para o problema parametrizado de cobertura de vértices, transformando a instância (G, k) de clique na instância (G', k') de cobertura de vértices. No entanto, esta transformação não é paramétrica, uma vez que $k' = n - k$ não é uma função somente em k. Na verdade, há indícios da não existência de uma transformação paramétrica nesta direção entre estes dois problemas, já que o problema de clique não pertence a FPT. Já a transformação paramétrica na direção contrária existe, uma vez que cobertura de vértices está em FPT.

Um esquema das transformações citadas acima é ilustrado na Figura 2.1.

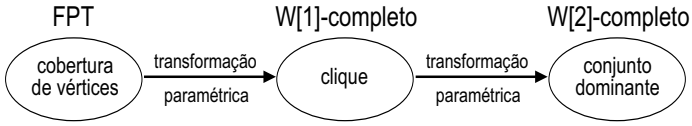

Figura 2.1: Transformações paramétricas aplicadas a alguns problemas parametrizados.

A propriedade essencial das transformações paramétricas é que, se L pode ser transformada em L' e $L' \in FPT$, então $L \in FPT$. Caso L seja $W[t]$-completo e $L' \in W[t]$ $(t \geq 1)$, onde W[t] é uma família da hierarquia W (abordada na seção 2.3), então podemos concluir que L' também é $W[t]-$completo.

2.2 Técnicas para prova de tratabilidade

Nesta seção, serão apresentadas duas técnicas elementares e amplamente aplicadas para se provar a tratabilidade por parâmetro fixo de determinado problema. Em vários casos, uma combinação das técnicas se faz necessária. Exemplos mais elaborados de ambas serão vistos nos próximos capítulos.

2.2.1 O método de busca em árvore

Esta técnica será demonstrada através de sua aplicação ao problema parametrizado previamente definido de cobertura de vértices. A tratabilidade deste problema é garantida pelo seguinte teorema.

Teorema 2.4 [7] *O problema parametrizado de cobertura de vértices pode ser resolvido em um tempo de $O(2^k \cdot n)$, onde n é o número de vértices no grafo (e a constante oculta α é independente de n e de k).*

Prova: Para resolvermos este problema, construímos uma árvore binária de altura k da seguinte forma: A raiz da árvore é rotulada com o conjunto vazio e o grafo G. Uma aresta $(u, v) \in E$ é escolhida. Em qualquer cobertura de vértices V' de G, necessariamente $u \in V'$ ou $v \in V'$. Então, são criados dois filhos do nó raiz, um para cada possibilidade. Assim, o primeiro filho é rotulado com $\{u\}$ e $G - u$, e o segundo filho é rotulado com $\{v\}$ e $G - v$. Em cada rótulo, o conjunto de vértices representa uma possível cobertura de vértices e o grafo representa o que permanece em G para ser coberto. Em geral, para um nó rotulado com o conjunto de vértices S e o subgrafo H de G, nós escolhemos uma aresta $(u, v) \in E(H)$ e criamos os dois filhos rotulados: $S \cup \{u\}$ e $H - u$, e $S \cup \{v\}$ e $H - v$. Se um nó de altura máxima k foi criado na árvore e o grafo do rótulo deste nó não possui arestas, então a cobertura de vértices de cardinalidade máxima k foi encontrada. Claramente, não é necessário explorar a árvore além da altura k. □

Para ilustrarmos este problema, consideremos como instância o grafo G da Figura 2.2 e o parâmetro $k = 2$.

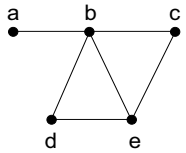

Figura 2.2: Grafo G fornecido como entrada para o problema parametrizado de cobertura de vértices.

Ao aplicarmos a técnica, como um possível resultado temos a árvore binária de busca rotulada da Figura 2.3.

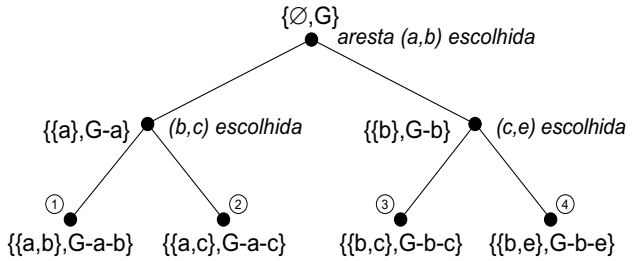

Figura 2.3: Árvore gerada pela execução do algoritmo aplicado ao grafo G.

A cada vez que o algoritmo chega a uma folha da árvore, o valor de k atingiu o valor 0. A única folha que representa uma solução é a apresentada em ④ , uma vez que, nas demais folhas, os grafos rotulados ainda possuem arestas. O conjunto de vértices $\{b, e\}$ constitui a cobertura de vértices alcançada em ④ .

2.2.2 O método de redução a um núcleo do problema

A idéia principal deste método é reduzir, em tempo polinomial, uma instância I do problema a uma instância equivalente I', tal que o tamanho de I' seja limitado por alguma função do parâmetro k independente da cardinalidade de I. Dessa forma, a instância I' constitui um núcleo do problema com cardinalidade constante. Logo, I' pode ser exaustivamente analisada, e sua solução pode ser apresentada como uma solução para I, caso exista.

Como exemplo, este método também será aplicado ao problema parametrizado de cobertura de vértices, a fim de prover uma comparação com o resultado da aplicação do método anterior ao mesmo problema. Obviamente, esta comparação é meramente ilustrativa. Nem sempre é possível aplicar ambas as técnicas a um determinado problema na mesma etapa da resolução. Além disso, os resultados de suas aplicações dependem das características do problema em questão.

Teorema 2.5 [7] *O problema parametrizado de cobertura de vértices pode ser resolvido em um tempo de* $O(n + k^k)$.

Prova: O algoritmo apresentado como prova é baseado na seguinte observação: Para um grafo G, todo vértice v de grau maior que k tem que pertencer a qualquer cobertura de vértices H de G, com $|H| \leq k$ (uma *k-cobertura* de vértices). Caso contrário, cada vizinho de v pertenceria à cobertura de vértices, excedendo o limite de k vértices. Os passos do algoritmo são descritos abaixo:

Passo 1: Localizar todos os vértices em G de grau maior que k. Seja p o número total desses vértices. Se $p > k$, então não existe uma k-cobertura de vértices para G. Caso contrário, seja $k' = k - p$.

Passo 2: Descartar os p vértices encontrados no *Passo 1* e as arestas incidentes a eles. Em seguida, retirar todos os vértices de grau 0, pois não faria sentido adicioná-los à cobertura de vértices. Se o grafo resultante G' tiver mais de $k'(k+1)$ vértices, não há solução.

Passo 3: Se G' não possuir uma k'-cobertura de vértices, não há solução. Caso contrário, qualquer k'-cobertura de vértices de G' unida aos p vértices do *Passo 1* formam uma k-cobertura de vértices de G.

O limite de $k'(k+1)$ no *Passo 2* é justificado pelo fato de que um grafo com uma k'-cobertura de vértices e graus limitados por k não podem possuir mais de $k'(k+1)$ vértices (k vizinhos de um vértice v pertencente à cobertura adicionado a ele próprio, multiplicado pelo total de vértices da cobertura). Os passos 1 e 2 podem ser realizados em um tempo de $O(n)$. Para um k fixo, o último passo é uma operação de tempo constante de $O(k^k)$, devido ao tamanho de G' que, no pior caso ($p = 0$ e nenhum vértice descartado no passo 2), será o grafo original G. □

Ao aplicarmos esta técnica ao exemplo da Figura 2.2 ($k = 2$), o *Passo 1* localiza os vértices b e e, resultando em $k' = 0$. O *Passo 2* retira esses vértices de G, bem como as arestas incidentes a eles, gerando o grafo resultante G' sem arestas. Em seguida, todos os demais vértices são retirados de G', por

possuirem grau 0. Como não há vértices restantes, o *Passo 3* conclui que os próprios vértices retirados no Passo 1 representam a solução.

2.3 Classes de complexidade parametrizada

Devido à necessidade de se construir uma teoria perfeita para classificar a aparente intratabilidade por parâmetro fixo de problemas como conjunto dominante, Downey e Fellows definiram classes apropriadas de problemas parametrizados, organizadas em uma hierarquia. Estas classes são baseadas intuitivamente na complexidade dos circuitos necessários para se verificar a validade de uma solução, ou alternativamente, a *profundidade lógica natural* do problema.

Primeiramente, são definidos os circuitos, formados por portas lógicas. Algumas delas possuem o tamanho da entrada limitado, enquanto outras possuem entrada irrestrita. Quanto à saída de uma porta lógica, assume-se que ela nunca é restrita.

Definição 2.6 [5, 7] *Um circuito booleano é de* tipo misto *se ele consistir de circuitos que possuam portas lógicas dos seguintes tipos:*

(1) *Portas lógicas estreitas:* portas *não*, *e* e *ou*, com tamanho da entrada limitado. Usualmente, assume-se que o tamanho limite da entrada é 2 para portas *e* e *ou*, e 1 para portas *não*.

(2) *Portas lógicas largas:* portas E e OU, com tamanho da entrada irrestrito.

A caixa baixa é usada para denotar portas lógicas estreitas (portas *ou* e *e*), enquanto que a caixa alta é usada para portas largas (E e OU).

A partir da definição de um circuito booleano, podemos definir sua profundidade lógica:

Definição 2.7 [5, 7] *A* profundidade *de um circuito C é definida pelo número máximo de portas lógicas (estreitas ou largas) em um caminho da entrada à saída em C. O* entrelaçamento *de um circuito C é o número máximo de portas lógicas largas em um caminho da entrada à saída de C.*

Dizemos que uma família de circuitos F possui *profundidade limitada* se existir uma constante h tal que todo circuito pertencente à família F possui profundidade máxima h. Dizemos que F possui *entrelaçamento limitado* se existir uma constante t tal que todo circuito em F possui entrelaçamento máximo t. Consideramos F *monótona* se os circuitos que a formam não possuem portas lógicas *não*. F é uma *família de circuitos de decisão* se cada circuito possui uma única saída. Dizemos que um circuito de decisão C *aceita* um vetor de entrada x se a saída do circuito possui valor 1 para x. O *peso* de vetor booleano x é o número de valores 1 no vetor.

Definição 2.8 [5, 7] *Seja F uma família de circuitos de decisão. Sabe-se que F pode conter muitos circuitos diferentes com um dado número de entradas. É associado a F o problema de circuito parametrizado $L_F = \{(C, k) : C$ aceita um vetor de entrada de peso $k\}$.*

Definição 2.9 [5, 7] *Um problema parametrizado L pertence a $W[t]$ ($W[t]$ monótona) se existir uma redução uniforme de L ao problema de circuito parametrizado L_F, para alguma família F de profundidade limitada, formada por circuitos de decisão do tipo misto (monótonos) de entrelaçamento máximo t.*

Conforme dito anteriormente, a classe de problemas tratáveis por parâmetro fixo é denotada por FPT. Assim, temos as relações de continência

$$FPT \subseteq W[1] \subseteq W[2] \subseteq \cdots$$

e a conjectura de que cada uma dessas relações é própria. A união das classes da hierarquia W é denotada por WH. Se $P = NP$, temos a seguinte implicação.

Lema 2.10 [7] *Se $P = NP$, então $WH \subseteq FPT$.*

O teorema a seguir representa um papel na teoria de complexidade parametrizada análogo ao desempenhado pelo Teorema de Cook [1] para NP-completude. Uma variação parametrizada de *satisfabilidade* baseada em uma

[1] A linguagem SAT de satisfabilidade de fórmulas booleanas é NP-completa.

forma normal para expressões booleanas provê os problemas identificados como completos para os vários níveis de WH. Para definir estes problemas, precisamos do seguinte conceito:

Definição 2.11 [7] *Uma expressão booleana X é dita* t-normalizada *se:*
(1) $t = 2$ *e X está na forma de produto de somas* (P-o-S),
(2) $t = 3$ *e X está na forma de produto de somas de produtos* (P-o-S-o-P),
(3) $t = 4$ *e X está na forma* P-o-S-o-P-o-S,
$\cdots$ *etc.*

A partir dele, obtemos o problema parametrizado que serve de ponto central para o teorema.

Satisfabilidade $t-$normalizada com peso associado
Instância: Uma expressão booleana $t-$normalizada X.
Parâmetro: Um inteiro positivo k.
Questão: X pode ser satisfeita por um vetor booleano de peso k?

Teorema 2.12 [5, 7] *A satisfabilidade $t-$normalizada com peso associado é completa para $W[t]$, $t \geq 2$.* □

A prova deste teorema pode ser encontrada em [5].

O problema parametrizado de conjunto independente é um exemplo de um problema completo para $W[1]$. Já conjunto dominante, é completo para $W[2]$. Para fazer referência à complexidade parametrizada de problemas cujas soluções podem ser verificadas em tempo *polinomial*, temos a definição da seguinte classe de complexidade.

Definição 2.13 [7] *Um problema parametrizado L pertence a $W[P]$ ($W[P]$* monótona*) se existir uma redução uniforme de L ao problema de circuito parametrizado L_F para alguma família de circuitos F.*

Podemos notar que $W[t]$ está contida em $W[P]$ para todo t, e que $W[P] = FPT$ se $P = NP$. Como exemplo, temos os seguintes problemas,

completos para $W[P]$:

Satisfabilidade de Circuito Monótono
Instância: Um circuito monótono C.
Parâmetro: Um inteiro positivo k.
Questão: C aceita um vetor de entrada de peso k?

Destruição de Subgrafos de Grau Três
Instância: Um grafo $G = (V, E)$.
Parâmetro: Um inteiro positivo k.
Questão: Existe um conjunto $X \subseteq V$ com no máximo k vértices tal que $G - X$ não possui nenhum subgrafo de grau mínimo três?

Capítulo 3

Modificação de grafos por propriedade hereditária

3.1 Introdução

Uma propriedade Π relativa a grafos é definida pelo conjunto de todos os grafos que a satisfaz, sendo qualquer grafo que satisfaça Π um Π-grafo. Por simplicação da linguagem, o conjunto dos grafos que satisfazem a propriedade também será chamado de Π. Se, para qualquer grafo $G \in \Pi$, todo subgrafo induzido de G for também um Π-grafo, então temos que Π é uma *propriedade hereditária* - o conjunto dos grafos cordais[1], por exemplo. Uma propriedade Π contém uma *caracterização por conjunto proibido* se existir um conjunto F de grafos tais que um grafo é um Π-grafo se e somente se não contiver nenhum grafo de F como subgrafo induzido. Se esse conjunto F for finito, então temos que Π contém uma *caracterização por conjunto proibido finito*. Como exemplo, podemos citar o conjunto dos grafos *linha*[2], cujo conjunto proibido aparece na Figura 3.1.

Sendo um *buraco* um ciclo sem corda com pelo menos 4 vértices, o conjunto dos grafos cordais, neste caso, não serve de exemplo, uma vez que seu conjunto proibido é infinito, pois possui infinitos buracos.

Para uma propriedade Π, o *problema geral de modificação* Π*-grafo* consiste em transformar um grafo G em um Π-grafo pela adição e deleção de

[1] Família de grafos caracterizados pela ausência de l-ciclos ($l > 3$) sem cordas.
[2] Um grafo linha é um grafo de interseção das arestas de outro grafo.

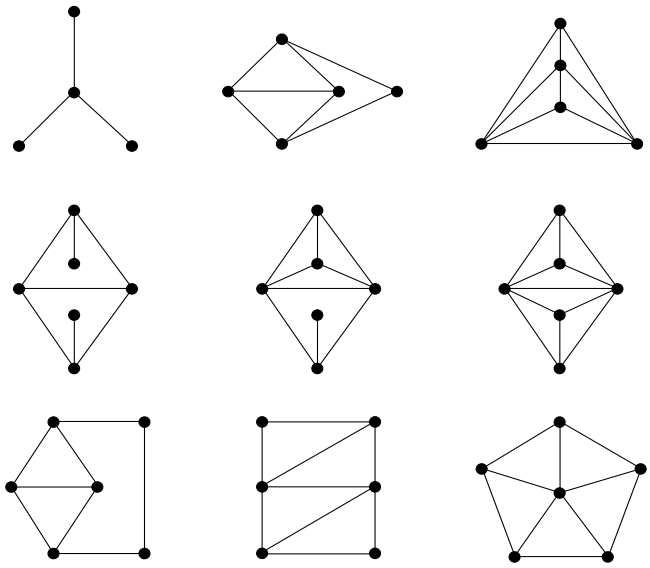

Figura 3.1: Conjunto proibido dos grafos *linha*.

arestas e vértices. Como exemplos típicos e bem estudados deste problema geral temos:

- O *problema de deleção de arestas*, ou *problema de subgrafo gerador máximo*: Encontrar um conjunto de arestas de cardinalidade mínima em G cuja deleção resulta em um Π-grafo.

- O *problema de deleção de vértices*, ou *problema de subgrafo induzido máximo*: Encontrar um conjunto de vértices de cardinalidade mínima em G cuja deleção resulta em um Π-grafo.

- O *problema de deleção de arestas e vértices*, ou *problema de subgrafo parcial máximo*: Encontrar um conjunto de arestas e vértices de cardinalidade mínima em G cuja deleção resulta em um Π-grafo.

- O *problema de adição de arestas*, ou *problema de supergrafo gerador mínimo*, ou *problema de complementação de grafo*: Encontrar um conjunto de novas arestas de cardinalidade mínima cuja adição em G resulta em um Π-grafo.

Abordaremos neste capítulo a versão parametrizada do problema de modificação Π-grafo para propriedades hereditárias - *problema de modificação* $\Pi(i,j,k)$*-grafo* (sendo i, j e k parâmetros fixos), onde E^C é o conjunto de arestas no complemento de G:

> Dado um grafo $G = (V, E)$, encontrar subconjuntos $V' \subseteq V$, $E' \subseteq E$ e $E^* \subseteq E^C$ com $|V'| \leq i$, $|E'| \leq j$ e $|E^*| \leq k$ tais que o grafo $G - V' - E' + E^*$ seja um Π-grafo.

Caso tal conjunto exista, G é dito ser um $\Pi(i,j,k)$-grafo, e o conjunto $V' \cup E' \cup E^*$ é um *modificador* de G. Como, para qualquer propriedade hereditária Π, nenhum Π-grafo pode ser gerado pela adição de vértices, não é necessário considerar a adição de vértices nesta abordagem do problema.

3.2 Propriedades de caracterizações por conjuntos proibidos finitos

Para estudar a tratabilidade por parâmetro fixo dos problemas relacionados às famílias de grafos caracterizadas por um conjunto proibido finito, é necessário, primeiramente, reportar-se ao fato de que uma propriedade Π é hereditária se e somente se ela possui uma caracterização por conjunto proibido ([2]).

Seja Π uma propriedade que possui uma caracterização por conjunto proibido finito. Seja v o número máximo de vértices entre todos os grafos do seu conjunto proibido F. Nos grafos *linha*, por exemplo, temos $v = 6$. Então, dado um grafo G com n vértices, poderíamos determinar se G é um Π-grafo em tempo de $O(n^v)$, pelo método de busca exaustiva. Para uma dada propriedade Π, o problema de modificação $\Pi(i,j,k)$-grafo poderia ser resolvido em tempo de $O(n^{i+2j+2k+v})$, considerando-se todas as possíveis formas de deletar no máximo i vértices e j arestas, e adicionar no máximo k arestas, verificando se cada grafo resultante é um Π-grafo. Logicamente, tal algoritmo não seria pertencente a FPT. No algoritmo proposto por Cai, a idéia é encontrar um subgrafo H de G, induzido minimal proibido (ou simplesmente subgrafo proibido) em relação a Π, e então destruí-lo pela deleção

de arestas e vértices e adição de arestas. Para garantir a complexidade, é necessário encontrar tal subgrafo em tempo polinomial, o que é garantido pelo seguinte teorema:

Teorema 3.1 [2] *Para qualquer propriedade hereditária* Π*, se um* Π*-grafo é reconhecível em um tempo* $T(m,n)$*, então, para qualquer grafo* G *que não seja um* Π*-grafo, um subgrafo proibido em relação a* Π *pode ser encontrado em* G *em um tempo de* $O(nT(m,n))$*.*

Prova: Seja A um algoritmo de reconhecimento para Π, executável em um tempo $T(m,n)$. Então, uma chamada $A(G)$ de A em G retorna "verdadeiro" se G é um Π-grafo e "falso" em caso contrário. Como detalhado no algoritmo abaixo, podemos usar A como um oráculo para encontrar um subgrafo proibido em relação a Π em um grafo $G \notin \Pi$. A idéia é verificar, para cada vértice v de G, se $G - v$ é um Π-grafo. Se isso ocorrer, então v pertence a um subgrafo proibido, senão, $G - v$ necessariamente contém um subgrafo proibido e assim o procedimento continua considerando $G - v$ da mesma forma. No algoritmo, F é usado para reservar os vértices de um subgrafo proibido:

$F := \emptyset$;
$V := V(G)$;
enquanto $V \neq \emptyset$ **faça**
    arbitrariamente escolher um vértice $v \in V$;
    $V := V - \{v\}$;
    se $A(G - v)$ **então**
        $F := F \cup \{v\}$;
    senão $G := G - v$;
fim enquanto

Claramente, o algoritmo acima termina sua execução após n iterações do loop "enquanto". Seja F' o conjunto F após o término da execução. É necessário provar que $G' = G[F']$ é um subgrafo proibido em relação a Π em G:

Seja v' o último vértice deletado de G, e G^* o grafo imediatamente antes da deleção do vértice v'. Então $G' = G^* - v'$. Como v' é deletado de G^*, G' não é um Π-grafo. Falta mostrar que G' é minimal no que diz respeito aos subgrafos induzidos proibidos, ou seja, para qualquer $u \in F'$, $G' - u$ é um Π-grafo. Supondo, por contradição, que exista um vértice $x \in F'$ tal que $G' - x$ não seja um Π-grafo. Seja G'' um grafo quando o vértice x está sob consideração. Então G' é um subgrafo induzido de G'', e assim $G'' - x$ não é um Π-grafo, já que $G' - x$ é um subgrafo induzido de $G'' - x$ e Π é hereditária. Isto implica que x teria sido deletado de G'', isto é, $x \notin F'$, uma contradição. Conseqüentemente, G' é um subgrafo de G proibido em relação a Π, provando a corretude do algoritmo.

Para a complexidade de tempo do algoritmo, temos que o oráculo A é chamado n vezes, cada uma levando no máximo um tempo $T(m, n)$, gerando uma complexidade total de tempo de $O(nT(m, n))$. □

Como exemplo, consideremos o conjunto dos grafos cordais e o grafo G não cordal da Figura 3.2.

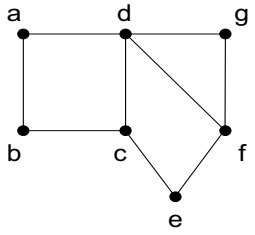

Figura 3.2: Exemplo de grafo não cordal.

Aplicando o algoritmo em G temos como início $F = \emptyset$ e $V = \{a, b, c, d, e, f, g\}$. Tomando como ordem de escolha dos vértices de V sua ordenação lexicográfica, temos as seguintes iterações:

1ª: vértice a escolhido
$V = \{b, c, d, e, f, g\}$
Como $G - a$ não é cordal, temos a exclusão do vértice a de G, demonstrada na Figura 3.3.

2ª: vértice b escolhido

$V = \{c, d, e, f, g\}$

Como $G - b$ não é cordal, temos a exclusão do vértice b de G, demonstrada na Figura 3.4.

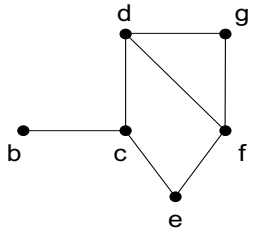

Figura 3.3: $G = G - a$.

3ª: vértice c escolhido

$V = \{d, e, f, g\}$

Como $G - c$ é cordal, temos $F = \{c\}$.

4ª: vértice d escolhido

$V = \{e, f, g\}$

Como $G - d$ é cordal, temos $F = \{c, d\}$.

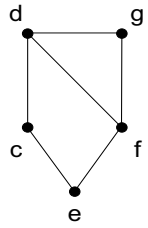

Figura 3.4: $G = G - b$.

5ª: vértice e escolhido

$V = \{f, g\}$

Como $G - e$ é cordal, temos $F = \{c, d, e\}$.

6ª: vértice f escolhido

$V = \{g\}$

Como $G - f$ é cordal, temos $F = \{c, d, e, f\}$.

7ª: vértice g escolhido

$V = \emptyset$

Como $G - g$ não é cordal, temos a exclusão do vértice g de G, demonstrada na Figura 3.5.

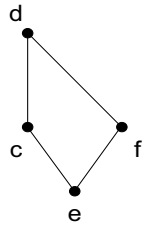

Figura 3.5: $G = G - g$.

Podemos notar que o grafo G final é exatamente o subgrafo induzido dos vértices reservados em F.

O tempo necessário para o reconhecimento de um grafo cordal é de $O(m+n)$ (abordado em detalhes no próximo capítulo). Logo, o tempo de execução para este exemplo é de $O(n^3)$.

De posse do teorema anterior, pode-se provar o seguinte resultado:

Teorema 3.2 [2] *O problema de modificação $\Pi(i,j,k)$-grafo $\in FPT$ para qualquer propriedade hereditária Π que admita uma caracterização por conjunto proibido finito.*

Prova: Para resolver o problema de modificação $\Pi(i,j,k)$-grafo, os dois passos a seguir são repetidos até que tenhamos chegado a um Π-grafo, ou tenhamos removido mais de i vértices e j arestas, e adicionado mais de k arestas. Se o primeiro caso ocorrer, temos que G é um $\Pi(i,j,k)$-grafo, o que será falso em caso contrário.

Passo 1: Encontrar um subgrafo H de G proibido em relação a Π.

Passo 2: Modificar G, deletando uma aresta ou um vértice de H, ou adicionando uma aresta a H.

A corretude do algoritmo é óbvia. Se G for um $\Pi(i,j,k)$-grafo, um modificador de G pode ser facilmente obtido comparando-se o grafo resultante com o grafo original.

Para estimarmos a complexidade de tempo do algoritmo, precisamos relembrar que Π é reconhecível em um tempo de $O(n^v)$. Assim, pelo Teorema 3.1, podemos encontrar em G um subgrafo proibido em relação a Π em um tempo de $O(n^{v+1})$. Como v é o número máximo de vértices de qualquer subgrafo proibido em relação a Π, existem no máximo $\binom{v}{2}$ maneiras de se adicionar ou deletar uma aresta de H, e no máximo v diferentes maneiras de deletar um vértice de H. Conseqüentemente, temos que o número total de grafos gerados pelo procedimento acima é no máximo $\binom{v}{2}^{j+k} v^i = O(v^{i+2j+2k})$. Logo, o tempo total de execução do algoritmo é de $O(v^{i+2j+2k} n^{v+1})$, o que implica que o algoritmo é uniformemente polinomial, já que v, i, j e k são constantes independentes de m e n. □

Retornando ao exemplo do conjunto Π dos grafos linha, seja o grafo G da Figura 3.6, para o qual desejamos resolver o problema de modificação $\Pi(1,1,0)$-grafo (retirar no máximo 1 vértice e 1 aresta, sem acrescentar arestas).

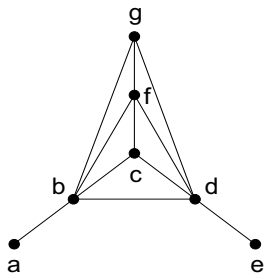

Figura 3.6: Exemplo de um grafo não linha.

Claramente, $G \notin \Pi$, já que possui os seguintes subgrafos proibidos: $\{b,c,d,f,g\}$, $\{a,b,c,g\}$ e $\{c,d,e,g\}$.

Aplicando o "*Passo 1*" da prova do Teorema 3.2, temos inicialmente $F = \emptyset$ e $V = \{a,b,c,d,e,f,g\}$. Tomando como ordem de escolha dos vértices de V sua ordenação lexicográfica, temos as seguintes iterações:

1ª: vértice a escolhido
$V = \{b,c,d,e,f,g\}$
Como $G - a$ não é linha, temos a exclusão do vértice a de G, demonstrada na Figura 3.7.

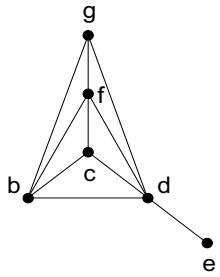

Figura 3.7: $G = G - a$.

2ª: vértice b escolhido
$V = \{c,d,e,f,g\}$
Como $G - b$ não é linha, temos a exclusão do vértice b de G (Figura 3.8).

3ª: vértice c escolhido
$V = \{d,e,f,g\}$
Como $G - c$ é um grafo linha, $F = \{c\}$.

4ª: vértice d escolhido
$V = \{e, f, g\}$
Como $G - d$ é um grafo linha, $F = \{c, d\}$.

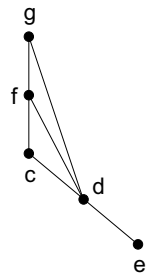

Figura 3.8: $G = G - b$.

5ª: vértice e escolhido
$V = \{f, g\}$
Como $G - e$ é um grafo linha,
$F = \{c, d, e\}$.

6ª: vértice f escolhido
$V = \{g\}$
Como $G - f$ não é linha, temos a
exclusão do vértice f de G,
demonstrada na Figura 3.9.

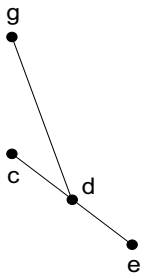

Figura 3.9: $G = G - f$.

7ª: vértice g escolhido
$V = \emptyset$
Como $G - g$ é um grafo linha, temos
$F = \{c, d, e, g\}$.

Assim, obtemos no fim do "*Passo 1*" o subgrafo proibido $F = \{c, d, e, g\}$. Como $|F| = 4$, temos no "*Passo 2*" 6 possibilidades de acrescentar ou retirar uma aresta (já que um grafo completo com 4 vértices possui 6 arestas) e 4 possibilidades de retirarmos um vértice de F. Nesta etapa temos, portanto, 10 grafos gerados a partir do grafo G original. Para continuação do exemplo, será escolhida a deleção do vértice e.

Voltando ao "*Passo 1*", este agora terá como entrada o grafo G' da Figura 3.10.

Temos no início $F = \emptyset$ e $V = \{a, b, c, d, f, g\}$. Tomando como ordem de escolha dos vértices de V sua ordenação lexicográfica, obtemos ao final das 6

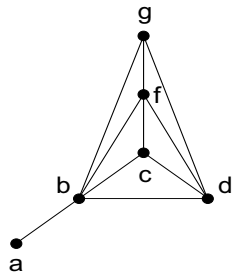

Figura 3.10: Grafo G' obtido de G.

iterações $F = \{b, c, d, f, g\}$. Dessa forma, temos 15 grafos gerados a partir de G', 10 pela adição ou deleção de uma aresta e 5 pela deleção de um vértice. Escolhendo a deleção da aresta (b, g), temos o grafo resultante G'' a seguir, que é um grafo linha.

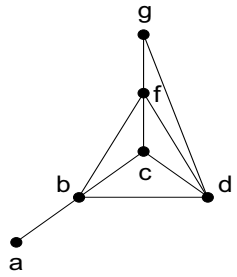

Figura 3.11: Grafo G'' obtido de G'.

Comparando o grafo resultante com o original, concluímos que o vértice e juntamente com a aresta (b, g) formam um *modificador* para o problema de modificação $\Pi(1, 1, 0)$-grafo apresentado. Obviamente, as escolhas apresentadas foram propositais para chegarmos a uma solução do problema. A execução deste procedimento resulta na geração de grafos como o crescimento de uma árvore de grafos. Dado um nó x dessa árvore, que representa um grafo G_x, cada filho desse nó representa uma possibilidade de modificação do grafo G_x. Tomando como exemplo o problema proposto para o grafo da Figura 3.6, este estará na raiz da árvore, possuindo 6 filhos. Por sua vez, o filho que representa o grafo $G - e$ possuirá 15 filhos. Dessa forma, cada nível representa uma iteração do procedimento. Se existir uma solução para

o problema, esta aparecerá em uma folha da árvore. No entanto, nem todas as folhas necessariamente representam uma solução, já que determinadas escolhas podem levar à impossibilidade de se resolver o problema sem ultrapassar o limite máximo das constantes i, j e k do problema. Como a geração de cada grafo (representado por um nó na árvore) é um procedimento simples, executado em um tempo constante, a complexidade de tempo do algoritmo é delimitada pelo número de grafos gerados nessa árvore, multiplicado pelo tempo necessário para verificar se cada um deles é um Π-grafo, como provado no Teorema 3.2.

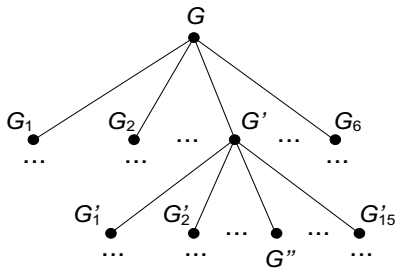

Figura 3.12: Geração de grafos pelo procedimento.

Consideremos agora Π como a família de todos os grafos cordais. Logo, Π é caracterizada por uma propriedade hereditária. Assim, o problema de cordalização(k) corresponde exatamente ao problema de modificação $\Pi(0, 0, k)$-grafo. Entretanto, o resultado geral obtido no Teorema 3.2 não pode ser aplicado a este problema, uma vez que seu conjunto proibido $F = \{H | H$ é um buraco$\}$ não é finito. No próximo capítulo, será mostrado que, apesar disso, o problema de cordalização(k) $\in FPT$.

Capítulo 4

O problema de cordalização(k)

4.1 Introdução

Um grafo G é chamado *cordal* ou *triangulado* se todo ciclo de comprimento maior que três em G possui uma corda. G é dito *completo* se, para todo par de vértices distintos (v_i, v_j), $v_i, v_j \in V(G)$, temos que $(v_i, v_j) \in E(G)$.

Seja G um grafo não cordal e F um conjunto de arestas entre os vértices de V de forma que $E \cap F = \emptyset$ e $G' = (V, E \cup F)$ é cordal. O conjunto F é chamado uma *cordalização* (*fill-in*) ou *triangulação* de G. Se $|F| \leq k$, então F é uma *k-cordalização* (ou *k-triangulação*) de G. Se não existir nenhum conjunto F' tal que $G' = (V, E \cup F')$ seja cordal e $|F'| < |F|$, então F é uma cordalização *mínima* de G. Denotamos por $\Phi(G)$ o tamanho da cordalização mínima de G.

O problema de cordalização mínima consiste em encontrar um conjunto mínimo de arestas tal que sua adição ao grafo original o torna cordal. A importância deste problema tem origem nas suas aplicações em álgebra numérica [13]. Em muitos campos, como simulação VLSI, solução de problemas lineares e processamento de sinais, uma eliminação gaussiana em uma matriz esparsa simétrica definida positiva tem que ser executada. Durante o processo de eliminação, elementos da matriz de entrada com valor zero podem se tornar não-zero. Diferentes ordens de eliminação podem introduzir diferentes conjuntos de novos elementos não-zero na matriz. O tempo de computação e o armazenamento necessários são dependentes do esparsamento da matriz. Além do mais, uma ordem de eliminação tal que um número mínimo

de elementos zero são preenchidos com não zero é desejável.

Segundo [12], Rose provou que o problema de encontrar uma ordem de eliminação para uma matriz simétrica positiva $n \times n$ $M = (m_{ij})$ tal que o mínimo de novos elementos não zero é introduzido é equivalente ao problema de cordalização mínima em um grafo não direcionado $G = (V, E)$ cujo vértice v_i corresponde à i-ésima linha e i-ésima coluna de M, na qual $(v_i, v_j) \in E$ se e somente se $M_{i,j} \neq 0$.

Devido à sua importância, o problema de cordalização mínima tem sido estudado intensivamente desde 1970. Sua NP-completude foi conjecturada em 1976 [14], permanecendo como problema aberto até 1981, quando Yannakakis [16] provou que computar a cordalização mínima de um dado grafo é um problema NP-completo. Assim, os pesquisadores deste problema guiaram-se considerando a cordalização mínima sob as seguintes perspectivas [1]:

1. Encontrar uma triangulação minimal pela relaxação do problema, o que pode ser feito em um tempo polinomial.

2. Descobrir se o problema é tratável por parâmetro fixo, o que foi provado por Cai [2] e, independentemente, por Kaplan, Shamir e Tarjan [12] (prova que será vista neste capítulo). Nessa abordagem do problema, estamos interessados em saber se o grafo $G = (V, E)$ pode ser transformado em um grafo cordal pela adição de no máximo k arestas (sendo k uma constante inteira) e, caso a resposta seja afirmativa, encontrar um conjunto F de arestas, $|F| \leq k$, tal que $G = (V, E \cup F)$ seja um grafo cordal.

3. Descobrir se o problema pode ser aproximado por um algoritmo polinomial. Em 2000, A. Natanzon, R Shamir e R. Sharan [13] publicaram o primeiro algoritmo aproximativo polinomial para o problema de cordalização, baseado no algoritmo desenvolvido em [12]. Sendo $\Phi(G)$ o tamanho da cordalização mínima do grafo G de entrada, é garantido que o algoritmo encontra uma solução de tamanho máximo $8(\Phi(G))^2$.

4.2 Caracterizando grafos cordais

Um vértice x de G é chamado *simplicial* se seu conjunto de vértices adjacentes $adj(x)$ induz um subgrafo completo de G, ou seja, o conjunto $adj(x)$ forma uma *clique* (não necessariamente maximal).

Dirac (em 1961) e posteriormente Lekkerkerker e Boland (1962) provaram que grafos cordais têm sempre um vértice simplicial (na verdade contém pelo menos dois deles). Baseados nisto (segundo [10, cap. 4]), e no fato de que a cordalidade de um grafo é uma propriedade hereditária, Fulkerson e Gross (em 1965) sugeriram um procedimento iterativo para o reconhecimento de grafos cordais. Este procedimento localiza repetidamente um vértice simplicial e o elimina do grafo, até que nenhum vértice permaneça (sendo assim o grafo original cordal) ou até que, em algum estágio, não exista nenhum vértice simplicial (concluindo então que o grafo original não é cordal). Para provar a corretude deste procedimento no próximo teorema, seguem alguns conceitos.

Seja $G = (V, E)$ um grafo não direcionado, e $\sigma = [v_1, v_2, \ldots, v_n]$ uma ordenação de seus vértices. Dizemos que σ é um *esquema perfeito de eliminação de vértices* (ou *ordenação perfeita*) se cada v_i é um vértice simplicial do subgrafo induzido $G_{\{v_i,\ldots,v_n\}}$. Ou seja, cada conjunto

$$X_i = \{v_j \in adj(v_i) | j > i\}$$

induz um subgrafo completo. Por exemplo, o grafo G_1 da Figura 4.1 possui um esquema perfeito de eliminação $\sigma = [a, g, b, f, c, e, d]$. Podemos facilmente notar que este esquema não é único. Na verdade, o grafo G_1 possui 96 esquemas perfeitos de eliminação diferentes. Em contrapartida, o grafo G_2 não possui vértice simplicial, logo não existe nenhuma ordenação perfeita.

Um subconjunto $S \subset V$ é um *separador de vértices* para vértices a e b não adjacentes (ou *a-b separador*) se a remoção de S do grafo separa a e b em duas componentes conexas distintas. Se nenhum subconjunto próprio de S for um *a-b* separador, então S é um *separador de vértices minimal* de a e b. Por exemplo, no grafo G_2 da Figura 4.1, o conjunto $\{y, z\}$ é um separador

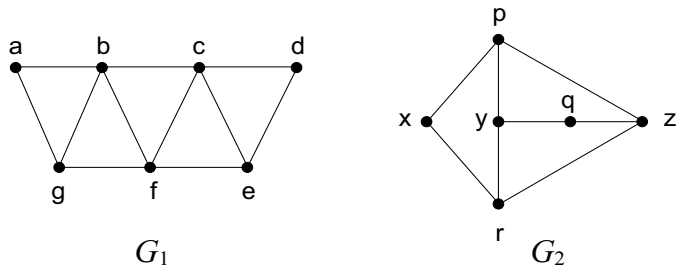

Figura 4.1: Dois grafos: um cordal (G_1) e um não cordal (G_2).

minimal de p e q, e o conjunto $\{x, y, z\}$ é um separador minimal de p e r. Em G_1, temos que todo separador minimal possui cardinalidade 2, sendo esses dois vértices adjacentes. Este fenômeno na verdade ocorre em todos os grafos cordais, como mostrado no Teorema 4.1.

Teorema 4.1 [10] *Seja G um grafo não direcionado. As seguintes afirmativas são equivalentes:*

(i) G é cordal.

(ii) G possui um esquema perfeito de eliminação. Além do mais, qualquer vértice simplicial pode iniciar o esquema de eliminação.

(iii) Qualquer separador de vértices minimal induz um subgrafo completo de G.

Prova: $(iii) \Rightarrow (i)$ Seja $[a, x, b, y_1, y_2, \ldots, y_k, a]$ $(k \geq 1)$ um ciclo simples de $G = (V, E)$. Qualquer $a - b$ separador minimal necessariamente contém os vértices x e y_i, para algum i. Logo, $(x, y_i) \in E$, sendo uma corda do ciclo.
$(i) \Rightarrow (iii)$ Seja S um $a - b$ separador minimal, e G_A e G_B as componentes conexas de G_{V-S} que contém a e b, respectivamente. Como S é minimal, cada vértice $x \in S$ é adjacente a algum vértice em A e a algum vértice em B. Para qualquer aresta $(x, y) \in S$, existem os caminhos $[x, a_1, \ldots, a_r, y]$ e $[y, b_1, \ldots, b_t, x]$, onde cada $a_i \in A$ e $b_i \in B$, de tal forma que esses caminhos possuem os menores comprimentos possíveis. Então, temos que $[x, a_1, \ldots, a_r, y, b_1, \ldots, b_t, x]$ é um ciclo simples de comprimento pelo menos

4, implicando que ele necessariamente tem uma corda. Pela definição de separador de vétices, temos que $(a_i, b_j) \notin E$ e, pela minimalidade de r e t, temos que $(a_i, a_j) \notin E$ e $(b_i, b_j) \notin E$. Logo, a única corda possível é $(x, y) \in E$.

Para provar as demais implicações, precisamos do seguinte lema:

Lema 4.2 [10] *Todo grafo cordal $G = (V, E)$ possui um vértice simplicial. Além do mais, se G não é uma clique, então ele tem dois vértices simpliciais não adjacentes.*

Prova: Caso G seja um grafo completo, o resultado segue trivialmente. Assume-se que G possui dois vértices não adjacentes a e b e que o lema é verdadeiro para todos os grafos com menos vértices que G. Seja S um $a - b$ separador minimal, e G_A e G_B as componentes conexas de G_{V-S} que contém a e b, respectivamente. Por indução, ou o subgrafo G_{A+S} possui dois vértices simpliciais não adjacentes, sendo um deles em A (já que S induz um subgrafo completo), ou o próprio G_{A+S} é completo e qualquer vértice de A é simplicial em G_{A+S}. Além disso, como $adj(A) \subseteq A + S$, um vértice simplicial de G_{A+S} em A é simplicial também em G. Similarmente, B contém um vértice simplicial de G. Isto prova o lema. □

Continuando a prova do Teorema 4.1:
$(i) \Rightarrow (ii)$ Utilizando o lema, se G é cordal, então ele tem um vértice simplicial x. Como $G_{V-\{x\}}$ é cordal e com menos vértices que G, ele tem, por indução, uma ordenação perfeita. Acrescentanto o vértice x no início desse esquema, obtemos uma ordenação perfeita para G .
$(ii) \Rightarrow (i)$ Seja C um ciclo simples de G, e x o vértice de C com o menor índice no esquema perfeito. Como $|adj(x) \cap C| \geq 2$, o fato de x ser simplicial garante que existe uma corda em C. □

Podemos também criar ordenações (não perfeitas) para grafos não cordais. Para um vértice v, a *deficiência* $D(v)$ é o conjunto de arestas definido

por

$$D(v) = \{(x, y) | x \in adj(v), y \in adj(v), y \notin adj(x), x \neq y\}.$$

O grafo G_v obtido de G por

(i) adição de arestas para que todos os vértices em $adj(v)$ sejam adjacentes entre si, e

(ii) deleção de v e de suas arestas incidentes

é um *grafo v-eliminação* de G. Ou seja,

$$G_v = (V - \{v\}, E(V - \{v\}) \cup D(v)).$$

Para um grafo ordenado $G_\alpha = (V, E, \alpha)$ (onde α é uma ordenação dos vértices de G), o *processo de eliminação*

$$P(G_\alpha) = [G = G_0, G_1, G_2, \ldots, G_{n-1}]$$

é uma seqüência de grafos de eliminação definidos recursivamente por $G_0 = G$, $G_i = (G_{i-1})_{x_i}$ para $i = 1, 2, \ldots, n-1$. Se $G_i = (V_i, E_i)$ para $i = 0, 1, \ldots, n-1$, a *cordalização* $F(G_\alpha)$ é definida por

$$F(G_\alpha) = \bigcup_{i=1}^{n-1} \tau_i,$$

onde $\tau_i = D(x_i)$ em G_{i-1}, e o *grafo de eliminação* G^*_α é definido por

$$G^*_\alpha = (V, E \cup F(G_\alpha)).$$

Ou seja, em se tratando de um grafo não cordal, $F(G_\alpha)$ são exatamente as arestas que precisaríamos acrescentar ao grafo ao longo de todo o processo de eliminação para que, a cada passo i, o vértice v_{i_α} se tornasse simplicial imediatamente antes da sua eliminação. Logo, dado um grafo $G = (V, E)$, uma ordenação α de V é perfeita se $F(G_\alpha) = \emptyset$.

Como exemplo, no grafo G_2 da Figura 4.1, $\alpha_1 = \{x, q, z, p, y, r\}$ produz um $F(G_{\alpha_1}) = \{(p, r), (y, z)\}$, que é uma cordalização mínima. Considerando a ordenação $\alpha_2 = \{p, q, r, x, y, z\}$, teremos $F(G_{\alpha_2}) = \{(x, y), (x, z), (y, z)\}$,

que não é mínima nem minimal, uma vez que $\{(x,y),(y,z)\} \subset F(G_{\alpha_2})$ também cordaliza G.

Para computarmos as arestas em G^*_α, precisamos dos seguintes conceitos. As provas dos lemas podem ser encontradas em [14]:

Em G_α, o conjunto de vértices *monotonamente adjacente* de um vértice v é definido por

$$madj(v) = adj(v) \cap \{w \in V | \alpha^{-1}(v) < \alpha^{-1}(w)\},$$

sendo $\alpha^{-1}(z)$ o índice do vértice z na ordenação α.

A notação $v \rightarrow w$ significa que $w \in madj(v)$.

Lema 4.3 [14] *Seja $G_\alpha = (V, E, \alpha)$ um grafo ordenado. Então (v, w) é uma aresta de $G^*_\alpha = (V, E \cup F(G_\alpha))$ se e somente se existir um caminho $\mu = [v = v_1, v_2, \ldots, v_{k+1} = w]$ em G_α tal que*

$$\alpha^{-1}(v_i) < min(\alpha^{-1}(v), \alpha^{-1}(w)), \quad 2 \leq i \leq k.$$

Lema 4.4 [14] *Seja $G_\alpha = (V, E, \alpha)$ um grafo ordenado. Então $E \cup F(G_\alpha)$ é o menor conjunto E^* de arestas tal que $E \subseteq E^*$ e*

$$(v \rightarrow w \text{ em } E^*) \Rightarrow (m(v) = w \text{ ou } m(v) \rightarrow w \text{ em } E^*),$$

onde $m(v)$ é o vértice u com mínimo $\alpha^{-1}(u)$ tal que $v \rightarrow u$ em E^*.

O seguinte algoritmo usa o Lema 4.4 para computar as arestas em G^*_α para qualquer grafo ordenado $G_\alpha = (V, E, \alpha)$. Se $A(v) = \{w | v \rightarrow w \text{ em } G_\alpha\}$ para todo v quando o algoritmo inicia, então $A(v) = \{w | v \rightarrow w \text{ em } G^*_\alpha\}$ para todo v quando o algoritmo termina.

ALGORITMO FILL: **início**

loop: **para** $i := 1$ **até** $n - 1$ **faça início**

$v := \alpha(i)$;

$m(v) := \alpha(min\{\alpha^{-1}(w) | w \in A(v)\})$;

adicionar: **para** $w \in A(v)$ **faça se** $w \neq m(v)$ **então**
adicionar w a $A(m(v))$;
fim fim FILL;

Conforme [14], o algoritmo acima pode ser implementado em $O(n + m')$, onde m' é o número de arestas de G^*_α.

Muitos problemas reconhecidamente NP-completos para grafos arbitrários são tratáveis para grafos cordais. Gavril (em 1972) apresentou algoritmos eficientes para encontrar todas as cliques maximais, cliques máximas, coloração mínima, conjuntos independentes máximos e cobertura de clique mínima em grafos cordais. Assumindo que uma ordenação perfeita é dada, (segundo [14]) é fácil implementar os algoritmos de Gavril para serem executados em um tempo de $O(n + m)$. Várias classes importantes de grafos, como árvores, k-árvores e grafos de intervalo são cordais, e o algoritmo de reconhecimento para grafos cordais pode ser utilizado de forma eficiente para o reconhecimento de grafos de intervalo.

4.3 Reconhecendo grafos cordais

Analisando o Lema 4.2, podemos observar que, dado um grafo cordal $G = (V, E)$, ele na verdade nos fornece a possibilidade de escolhermos um vértice dentre pelo menos dois, para cada posição do esquema perfeito σ de G. Isso nos permite evitar a escolha de um determinado vértice v_n, salvando-o na última posição de σ. Similarmente, podemos escolher qualquer vértice v_{n-1} adjacente a v_n, e salvá-lo na $(n-1)$-ésima posição. Dessa forma, prosseguimos contruindo σ de trás para frente. Utilizando essa lógica, Rose, Tarjan e Lueker [14] apresentaram em 1976 um algoritmo (LEX BFS) com complexidade de tempo de $O(n + m)$ para encontrar ordenações perfeitas, usando um percurso lexicográfico (um tipo especial de percurso em largura).

Em um trabalho de 1976, Tarjan demonstrou outro método de percurso em grafos, também com tempo de $O(n + m)$, que pode ser utilizado para reconhecimento de grafos cordais, chamado *percurso de cardinalidade máxi-*

ma (MCS). Mais detalhes sobre este algoritmo, inclusive sua implementação, podem ser encontrados em [15].

4.3.1 Percurso em largura (BFS)

Dado um grafo G, um *percurso em largura* em G é um processo que se inicia em um vértice s e sistematicamente examina as arestas de G, usando o seguinte algoritmo:

ALGORITMO BFS: **início**
inicializar *fila* vazia;
$nível(s) := 0$;
marcar s como explorado;
inserir s em *fila*;
enquanto *fila* não estiver vazia **faça início**
remover o primeiro vértice v da *fila*;
explorar: **para** cada $w \in adj(v)$ **faça se** w é não explorado **então início**
adicionar w no fim da *fila*;
$nível(w) := nível(v) + 1$;
marcar (v, w) como uma aresta de árvore;
marcar w como explorado;
fim fim fim BFS;

A cada passo, o algoritmo escolhe uma aresta incidente ao vértice alcançado há mais tempo. O vértice da outra ponta da aresta pode ser um vértice que também já foi alcançado ou um novo vértice, que passará a ser considerado alcançado.

Durante a execução deste algoritmo, o procedimento *explorar* visita cada aresta exatamente duas vezes; uma para $w \in adj(v)$ e outra para $v \in adj(w)$. Ao executarmos este algoritmo sobre um grafo G, temos como resultado:

(i) a criação de uma árvore geradora de G, dada pelas arestas (v, w) tais que w ainda é não-visitado quando o procedimento *explorar* é executado com $w \in adj(v)$, e

(ii) o particionamento dos vértices de G em *níveis*: sendo v um vértice, $nível(v) = i$ se o menor caminho de s até v possui comprimento i.

Podemos observar que a escolha da aresta a ser explorada é arbitrária. Conseqüentemente, o algoritmo BFS é sensível à ordenação dos vértices na lista de adjacências.

4.3.2 Percurso lexicográfico em largura (LEX BFS)

Na versão lexicográfica do BFS, aplicada ao reconhecimento de grafos cordais, os vértices do grafo de entrada são numerados de n até 1. Durante o percurso, a cada vértice v é associado um *rótulo* formado por um conjunto de números selecionados de $\{1, 2, \ldots, n\}$, ordenados de forma decrescente. Dados dois rótulos, $L_1 = [p_1, p_2, \ldots, p_k]$ e $L_2 = [q_1, q_2, \ldots, q_l]$, temos que $L_1 < L_2$ se, para algum j, $p_i = q_i$ para $i = 1, 2, \ldots, j-1$ e $p_j < q_j$, ou se $p_i = q_i$ para $i = 1, 2, \ldots, k$ e $k < l$. $L_1 = L_2$ se $k = l$ e $p_i = q_i$, $1 \leq i \leq k$.

• Gerando ordenações que implicam em triangulações minimais (*ordenações minimais*)

Considere o seguinte esquema de ordenação:

ALGORITMO LEX M: **início**
atribua o rótulo $\emptyset$ a todos os vértices;
para $i := n$ **até** 1 **faça início**
select: escolha um vértice não numerado v com maior *rótulo*;
comentário atribua a v o número i;
$\alpha(i) := v$;
update: **para** cada vértice não numerado w tal que existe um caminho $[v = v_1, v_2, \ldots, v_{k+1} = w]$ com v_j não numerado e $rótulo(v_j) < rótulo(w)$ para $j = 2, 3, \ldots, k$ **faça**
adicionar i ao $rótulo(w)$;
fim fim LEX M;

Este algoritmo contrói uma ordenação α para um grafo $G = (V, E)$ inicialmente não ordenado e constrói um rótulo $L(v)$ com o valor final de *rótulo*(v) para cada $v \in V$. Uma ordenação gerada por este algoritmo é chamada *ordenação lexicográfica*. A condição complicada do procedimento *update* é necessária porque o grafo G de entrada pode não ser cordal, já que o objetivo do algoritmo é encontrar uma ordenação minimal, e não perfeita (pois esta pode não existir).

Para estabelecermos a corretude do algoritmo, precisamos de alguns resultados, cujas provas podem ser encontradas em [14]:

Teorema 4.5 [14] *Seja $G = (V, E)$ um grafo e $G' = (V, E \cup F)$ cordal. Então F é um triangulação minimal se e somente se cada $f \in F$ é uma corda única de um 4-ciclo em G'.*

Lema 4.6 [14] *Seja $G = (V, E)$ um grafo com ordenação lexicográfica α e os rótulos $L(v)$, $v \in V$.*

(i) *Se $L_i(v) < L_i(w)$, então $L_j(v) < L_j(w)$ para todo $1 \leq j \leq i$.*

(ii) *$L_i(w) \leq L_j(w)$ para todo $j \leq i$.*

(iii) *Se $\alpha^{-1}(w) = j < \alpha^{-1}(v) = i$, então ou $L_i(w) < L_i(v)$ e $L(w) < L(v)$, ou $L_i(w) = L_i(v)$ e $L(v) \leq L(w)$.*

(iv) *Se $L(w) < L(v)$ com $\alpha^{-1}(w) = j$ e $\alpha^{-1}(v) = i$, então ou $j < i$ com $L_i(w) < L_i(v)$, ou $i < j$ com $L_j(w) = L_i(v)$.*

(v) *$j \in L(w)$ se e somente se $\alpha^{-1}(w) < j$ e existir um $v = \alpha(j), w$ caminho $[v = v_1, v_2, \ldots, v_{k+1} = w]$ tal que $L_j(v_i) < L_j(w)$ e $\alpha^{-1}(v_i) < j$, $2 \leq i \leq k$.*

As provas dos itens deste lema são diretas. As propriedades (i) e (ii) seguem das definições dos rótulos e de relação de ordem. (iii) e (iv) resumem o procedimento *select* do algoritmo. A propriedade (v) segue de (i), (ii) e do procedimento *update*, e significa que os rótulos produzidos atualizados por uma execução do procedimento *update* dependem somente dos rótulos antigos e não da ordem de atualização.

Lema 4.7 [14] *Se α é uma ordenação lexicográfica de $G = (V, E)$, então, em G^*_α, $L(w) = \{\alpha^{-1}(v) | w \rightarrow v\}$ para $w \in V$.*

Teorema 4.8 [14] *Seja $G = (V, E)$ um grafo com ordenação lexicográfica α e rótulos $L(v)$, $v \in V$. Então, qualquer aresta $(v, w) \in F(G_\alpha)$ é uma corda única de algum 4-ciclo $\mu = [p, v, q, w]$ em G^*_α.*

Os Teoremas 4.5 e 4.8 implicam imediatamente no seguinte.

Teorema 4.9 [14] *Seja $G = (V, E)$ um grafo com ordenação lexicográfica α. Então α é uma ordenação minimal.*

A implementação pode ser encontrada em [14] e requer tempo de execução de $O(nm)$ e espaço de $O(n + m)$.

- Gerando ordenações perfeitas

Como toda ordenação minimal de um grafo cordal é também perfeita, podemos testar G para ver se ele é perfeito gerando uma ordenação α usando o algoritmo LEX M e testando se $F(G_\alpha) = \emptyset$ usando FILL. Entretanto, há uma maneira melhor. Se G é perfeito e α é uma ordenação lexicográfica, então $F(G_\alpha) = \emptyset$ e, pelo Lema 4.7, $L(w) = \{\alpha^{-1}(v) | w \rightarrow v\}$ em G. Simplificando o procedimento *update* de Lex M, teremos:

ALGORITMO LEX BFS: **início**
atribua o rótulo $\emptyset$ a todos os vértices;
para $i := n$ **até** 1 **faça início**
select: escolha um vértice não numerado v com maior *rótulo*;
comentário atribua a v o número i;
$\alpha(i) := v$;
update: **para** cada vértice não numerado $w \in adj(v)$ **faça**
adicionar i ao *rótulo*(w);
fim fim LEX BFS;

Desta forma, LEX BFS irá gerar uma ordenação α que, pela observação acima, será perfeita se G for cordal. Logo, podemos usar LEX BFS para gerar uma ordenação e então usar o algoritmo FILL para calcular sua cordalização. Se esta for vazia, então o grafo é cordal e a ordenação é perfeita. Caso contrário, o grafo não é cordal e a ordenação α obtida pode não ser minimal.

Como citado anteriormente, este algoritmo possui tempo de execução de $O(n + m)$, utilizando espaço de $O(n + m)$.

Para ilustrar a execução do algoritmo, consideremos o grafo G a seguir:

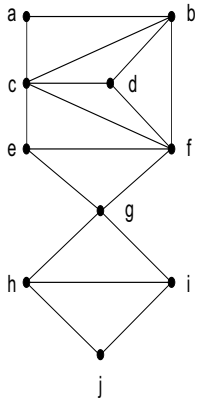

Figura 4.2: Grafo cordal G.

Sendo cordal, ele possui uma ordem perfeita de eliminação (não única). Aplicando o algoritmo, temos inicialmente todos os rótulos iguais a $\emptyset$ e nenhum vértice numerado. Assim, na 1ª iteração, podemos escolher qualquer vértice. Escolhendo o vértice g, temos as seguintes iterações:

1ª: vértice g escolhido dentre todos os demais:

vértices	*rótulos*	numeração
a	$\emptyset$	
b	$\emptyset$	
c	$\emptyset$	
d	$\emptyset$	
e	$\{10\}$	
f	$\{10\}$	
g	$\emptyset$	10
h	$\{10\}$	
i	$\{10\}$	
j	$\emptyset$	

2ª: vértice f escolhido dentre $\{e, f, h, i\}$:

vértices	*rótulos*	numeração
a	$\emptyset$	
b	$\{9\}$	
c	$\{9\}$	
d	$\{9\}$	
e	$\{10, 9\}$	
f	$\{10\}$	9
g	$\emptyset$	10
h	$\{10\}$	
i	$\{10\}$	
j	$\emptyset$	

3ª: vértice e selecionado:

vértices	*rótulos*	numeração
a	$\emptyset$	
b	$\{9\}$	
c	$\{9, 8\}$	
d	$\{9\}$	
e	$\{10, 9\}$	8
f	$\{10\}$	9
g	$\emptyset$	10
h	$\{10\}$	
i	$\{10\}$	
j	$\emptyset$	

4^a: vértice h escolhido dentre $\{h, i\}$:

vértices	*rótulos*	numeração
a	$\emptyset$	
b	$\{9\}$	
c	$\{9, 8\}$	
d	$\{9\}$	
e	$\{10, 9\}$	8
f	$\{10\}$	9
g	$\emptyset$	10
h	$\{10\}$	7
i	$\{10, 7\}$	
j	$\{7\}$	

5^a: vértice i selecionado:

vértices	*rótulos*	numeração
a	$\emptyset$	
b	$\{9\}$	
c	$\{9, 8\}$	
d	$\{9\}$	
e	$\{10, 9\}$	8
f	$\{10\}$	9
g	$\emptyset$	10
h	$\{10\}$	7
i	$\{10, 7\}$	6
j	$\{7, 6\}$	

6^a: vértice c selecionado:

vértices	*rótulos*	numeração
a	$\{5\}$	
b	$\{9, 5\}$	
c	$\{9, 8\}$	5
d	$\{9, 5\}$	
e	$\{10, 9\}$	8
f	$\{10\}$	9
g	$\emptyset$	10
h	$\{10\}$	7
i	$\{10, 7\}$	6
j	$\{7, 6\}$	

7^a: vértice d escolhido dentre $\{b, d\}$:

vértices	*rótulos*	numeração
a	$\{5\}$	
b	$\{9, 5, 4\}$	
c	$\{9, 8\}$	5
d	$\{9, 5\}$	4
e	$\{10, 9\}$	8
f	$\{10\}$	9
g	$\emptyset$	10
h	$\{10\}$	7
i	$\{10, 7\}$	6
j	$\{7, 6\}$	

8^a: vértice b selecionado:

vértices	*rótulos*	numeração
a	$\{5, 3\}$	
b	$\{9, 5, 4\}$	3
c	$\{9, 8\}$	5
d	$\{9, 5\}$	4
e	$\{10, 9\}$	8
f	$\{10\}$	9
g	$\emptyset$	10
h	$\{10\}$	7
i	$\{10, 7\}$	6
j	$\{7, 6\}$	

9^a: vértice j selecionado:

vértices	*rótulos*	numeração
a	$\{5, 3\}$	
b	$\{9, 5, 4\}$	3
c	$\{9, 8\}$	5
d	$\{9, 5\}$	4
e	$\{10, 9\}$	8
f	$\{10\}$	9
g	$\emptyset$	10
h	$\{10\}$	7
i	$\{10, 7\}$	6
j	$\{7, 6\}$	2

10^a: vértice a selecionado:

vértices	*rótulos*	numeração
a	$\{5,3\}$	1
b	$\{9,5,4\}$	3
c	$\{9,8\}$	5
d	$\{9,5\}$	4
e	$\{10,9\}$	8
f	$\{10\}$	9
g	$\emptyset$	10
h	$\{10\}$	7
i	$\{10,7\}$	6
j	$\{7,6\}$	2

Assim, encontramos ao fim da execução do algoritmo a ordenação perfeita $\alpha = \{a, j, b, d, c, i, h, e, f, g\}$ de G.

4.4 Tratabilidade do problema de cordalização(k)

Nesta seção apresentaremos o algoritmo desenvolvido em [12], como prova da tratabilidade do problema de cordalização(k).

4.4.1 Um algoritmo linear para k fixo

Primeiramente, precisamos limitar o número total de triangulações minimais para um dado ciclo C. Para tanto, consideremos o seguinte lema:

Lema 4.10 [12] *Existe uma correspondência 1-1 entre triangulações minimais de um ciclo com l vértices e árvores binárias com $l-2$ nós internos.*

Denotemos por c_l o l-ésimo *número de Catalão*, cuja fórmula é $c_l = \binom{2l}{l}\frac{1}{l+1}$. De acordo com [4, pág. 262], temos que

$$c_l = \frac{4^l}{\sqrt{\pi}\, l^{3/2}} \left(1 + O\left(\frac{1}{l}\right)\right).$$

A partir desta equação, podemos concluir que $c_l < 4^l$. Sendo b_n o número de árvores binárias com n nós internos, temos que b_n satisfaz a recorrência $b_0 = 1, b_n = \sum_{i+j=n-1} b_i b_j$, cuja solução, segundo [12], é $b_n = c_n$. Logo, temos como resultado:

Lema 4.11 [12] *O número de triangulações minimais de um l-ciclo é* $c_{l-2} \leq 4^{l-2}$.

O algoritmo desta seção irá atravessar parte de uma árvore de busca, em que cada nó corresponde a um supergrafo de G. A raiz dessa árvore é o próprio grafo G. Para gerar os filhos de um nó interno x, que corresponde a um grafo G', é necessário encontrar um buraco C em G'. O nó x terá um filho para cada triangulação minimal de C. O grafo correspondente a um filho de x é obtido adicionando-se a G' a triangulação minimal correspondente. Se $|C| = l$, então, pelo Lema 4.11, o nó x terá c_{l-2} filhos. Cada folha da árvore corresponde a um supergrafo cordal de G. Assim, cada triangulação minimal de G é representada por pelo menos uma folha.

Conforme dito anteriormente, um buraco pode ser encontrado em tempo de $O(m + n)$. Assim, segundo [12], podemos gerar todas as triangulações minimais em tempo de $O(|C|)$ por triangulação.

Na verdade, o algoritmo visita somente os nós da árvore de busca que correspondem a supergrafos de G com no máximo k arestas adicionais. Se algum nó dessa forma é uma folha, então uma k-triangulação de G foi encontrada. Caso contrário, esta não existe.

Teorema 4.12 [12] *Todas as k-triangulações minimais de um grafo G podem ser encontradas em tempo de* $O(2^{4k}m)$.

Prova: Seja T uma subárvore da árvore de busca atravessada pelo algoritmo. Para um nó $x \in T$, seja $G_x = (V, E_x)$ o supergrafo de G correspondente, d_x o comprimento máximo de um caminho de x até uma folha de T, e $a_x = \max(\{|E_f| - |E_x| \ / \ f \text{ é uma folha descendente de } x\})$. Denotemos por f_x o número total de folhas entre os descendentes de x. A título de ilustração, na Figura 4.3, temos $d_x = 3$ e $f_x = 5$.

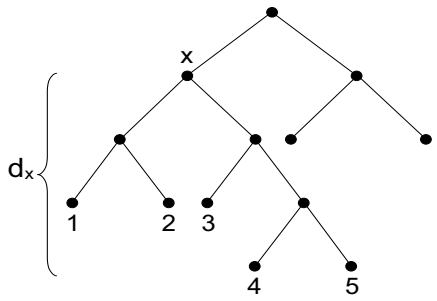

Figura 4.3: Exemplo de uma subárvore atravessada pelo algoritmo.

Consideremos a seguinte relação, provada por indução: $f_x \leq 4^{d_x+a_x}$. Assume-se que a relação é válida para todos os filhos de um nó x. Seja l o comprimento de um ciclo detectado em x. Sejam $d_{max} = \max\{d_y \mid y \text{ é um filho de } x\}$ e $a_{max} = \max\{a_y \mid y \text{ é um filho de } x\}$. Usando a hipótese de indução, o número de folhas de qualquer dos c_{l-2} filhos de x é limitado por $4^{d_{max}+a_{max}}$. Usando o Lema 4.11, temos que f_x é limitado por $4^{l-2}4^{d_{max}+a_{max}} = 4^{d_{max}+1+a_{max}+l-3}$. Como $d_x = d_{max} + 1$ e o tamanho de toda triangulação minimal de um l-ciclo é $l - 3$, temos que o número total de folhas descendentes de x é limitado por $4^{d_x+a_x}$.

Precisamos agora estimar o total de nós da árvore. É fácil ver que, no pior caso, teremos uma árvore cheia. Assim, considerando agora T como toda a árvore, temos:

$$\begin{aligned} \text{nº. nós}(T) &\leq 2f_x - 1 \\ &\leq 2 \cdot 4^{d_x+a_x} \end{aligned}$$

Para limitar d_x, devemos observar que a árvore de busca atingirá altura máxima quando o número de arestas acrescentadas a cada nível for mínimo. Assim, se todos os buracos encontrados desde a raiz até a folha mais distante forem de comprimento 4 (o menor possível), teremos apenas 1 aresta acrescentada a cada mudança de nível. Como o algoritmo procura por uma k-cordalização, temos $d_x \leq k$ e, pela mesma razão, obtemos $a_x \leq k$.

Substituindo, obtemos que o número máximo de nós é:

$$\text{nº. nós}(T) \leq 2 \cdot 4^{2k}$$

Para cada nó da árvore, é necessário encontrar um buraco e gerar os filhos correspondentes a cada triangulação minimal. Como visto anteriormente, isso pode ser feito em tempo de $O(n+m)$. Como $m = O(n^2)$, temos que o tempo total de execução do algoritmo é de $O(2^{4k}m)$. □

Na verdade, o algoritmo pode listar a mesma k-triangulação (caso exista) várias vezes. Esta redundância pode ser eliminada armazenando-se as soluções e checando-se a cada solução obtida se esta já havia sido encontrada antes.

4.4.2 Um algoritmo com fator multiplicativo polinomial

Nesta seção será descrito o algoritmo proposto com tempo de execução de $O(k^2nm + f(k))$.

Caso o grafo G dado possa ser cordalizado com no máximo k arestas, este algoritmo particiona o conjunto de vértices de G em dois subconjuntos A e B, tais que o tamanho de A é de $O(k^3)$ e não existem buracos em G que contenham vértices em B.

Particionando o grafo

O algoritmo utiliza três procedimentos principais, denominados P_1, P_2 e P_3, executados em seqüência:

(P_1) *Extraindo buracos independentes:*

Este procedimento inicia com $B = V$ e $A = \emptyset$. Repetidamente, ele encontra um buraco em G_B, usando o algoritmo MCS (seção 4.3), por exemplo, e move seus vértices para A. Assim, quando P_1 termina, o subgrafo induzido por B é cordal.

O algoritmo mantém um contador (de limite inferior) dinâmico chamado cc, que armazena o número mínimo de cordas necessárias para triangularizar G. Após detectar um ciclo sem corda C_i, cc é incrementado de $|C_i| - 3$. Se, em algum ponto da execução do algoritmo, tivermos $cc > k$, este pára com resposta negativa. Por outro lado, P_1 termina quando não houver mais buracos em B e $cc = \sum_{i=1}^{j}(|C_i| - 3) \leq k$.

A complexidade de tempo deste procedimento é de $O(km)$, uma vez que a detecção do ciclo ocorre em tempo linear e o número de ciclos detectados é de no máximo k, já que cada um deles é responsável pelo acréscimo de pelo menos 1 aresta. O tamanho do conjunto A após a execução deste procedimento é de $O(k)$.

(P_2) *Extraindo buracos conexos com caminhos independentes:*

Este procedimento procura por buracos que atravessam os conjuntos A e B, e contêm pelo menos dois vértices consecutivos em B. Seja C este ciclo, $|C| = l$. Se $l > k+3$, o algoritmo pára com resposta negativa. Caso contrário, todo subcaminho maximal de C contendo somente vértices de B é movido para A se seu tamanho for de pelo menos um (ou seja, se contiver pelo menos dois vértices). O incremento de cc dependerá da estrutura do ciclo C. Para especificarmos este incremento, precisamos do seguinte lema:

Lema 4.13 [12] *Sejam C um buraco e p um caminho em C de comprimento l tal que $1 \leq l \leq |C|-2$. Se $l = |C|-2$, então, em toda triangulação minimal de C, existem pelo menos $l-1$ cordas incidentes a pelo menos um vértice de p. Se $l < |C| - 2$, então, em toda triangulação minimal de C, existem pelo menos l cordas incidentes a pelo menos um vértice em p.*

Prova: Se $l = |C| - 2$, então há somente um vértice de C que não pertence a p. Logo, toda corda em uma triangulação minimal de C será incidente a algum vértice de p. Como neste caso o buraco possui comprimento $l + 2$, teremos $l + 2 - 3 = l - 1$ cordas incidentes a p.

Se $l < |C| - 2$, então a prova se dá por indução no comprimento do caminho. Como caso base, temos que o lema obviamente é válido para $l = 1$

(comprimento 1), já que é necessário haver uma corda incidente a pelo menos um dos vértices em p. A Figura 4.4 ilustra essa situação.

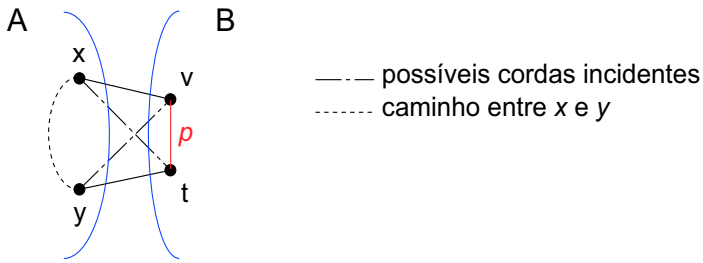

Figura 4.4: Caso base para prova do Lema 4.13.

Assume-se que o lema seja verdadeiro para todo caminho com comprimento menor que l. Sejam p um caminho com comprimento l e (a, b) uma corda incidente a p, que divide o ciclo C em dois ciclos, C_1 e C_2. Temos dois casos:

Caso 1: $a, b \in p$. Seja l_1 o comprimento do subcaminho de p que conecta a e b. Sem perda de generalidade, podemos assumir que $l_1 = |C_1| - 1$. Seja p' o caminho entre os extremos de p passando através de (a, b) em C_2, e seja $l_2 = |p'|$. Existirão pelo menos $l_1 - 2$ cordas incidentes a p em C_1, e, por hipótese indução, l_2 cordas incidentes a p' em C_2. Assim, o número total de cordas incidentes a p será pelo menos $(l_1 - 2) + l_2 + 1 = l$, sendo a aresta adicionada ao final da equação correspondente à aresta (a, b). A Figura 4.5 demonstra um exemplo desse caso.

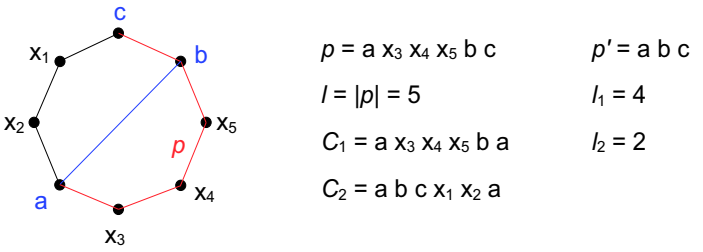

Figura 4.5: Caso 1 do Lema 4.13.

Caso 2: $a \in p$, $b \notin p$. Sejam $p_1 = p \cap C_1$ e $p_2 = p \cap C_2$. Para pelo menos um p_i, $i = 1, 2$, temos $|p_i| < |C_i| - 2$. Isso acontece porque, como $l < |C| - 2$, teremos sempre pelo menos dois vértices não pertencentes a p. Um deles é o próprio vértice b. O outro pertencerá a um dos dois ciclos, gerando nesse ciclo uma seqüência de pelo menos 3 arestas não pertencentes a p. Sem perda de generalidade, seja $|p_1| < |C_1| - 2$. Aplicando a hipótese de indução em C_1 e usando a parte anterior desse lema em C_2, conclui-se que o número total de cordas incidentes a p é pelo menos $l_1 + (l_2 - 1) + 1 = l$. A Figura 4.6 ilustra um exemplo desse caso. □

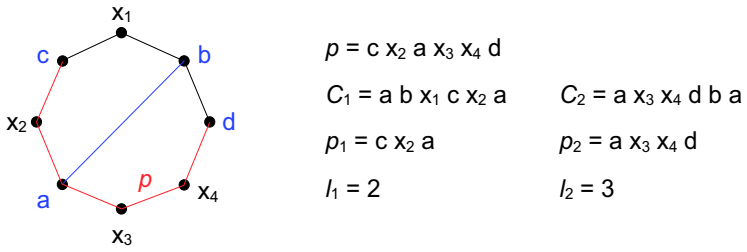

Figura 4.6: Caso 2 do Lema 4.13.

Suponhamos que C seja um l-ciclo sem corda que contenha $j \geq 1$ subcaminhos maximais disjuntos $p_1, p_2, \cdots, p_j$, cada um de comprimento pelo menos um, que estão em B. Seja $l_i = |p_i|$, $i = 1, \cdots, j$. Obviamente, se $l_1 = l - 2$, então $j = 1$, ou seja, existe somente um subcaminho. Caso contrário, temos $l_i < l - 2$, para todo $1 \leq i \leq j$. Usando o lema anterior, incrementa-se cc da seguinte maneira: Se existir um único subcaminho, cc é incrementado ou de $(l_1 - 1)$, se $l_1 = l - 2$, ou de l_1, se $l_1 < l - 2$. Caso contrário, cc é incrementado pelo maior valor entre $\frac{1}{2}\sum_{i=1}^{j} l_i$ (o fator $\frac{1}{2}$ é necessário porque uma corda pode ser contada duas vezes no somatório, uma vez que ambos os extremos dessa corda podem ser vértices de algum p_i) e $\max\{l_i \mid 1 \leq i \leq j\}$. Na Figura 4.7, por exemplo, temos $\frac{1}{2}\sum_{i=1}^{j} l_i = 4 < l_3$. Neste caso, cc seria incrementado de 5.

P_2 termina quando cc é maior que k ou quando não existem mais ciclos desse tipo. Para especificarmos como detectar um buraco C com vértices consecutivos em B, precisamos da seguinte observação.

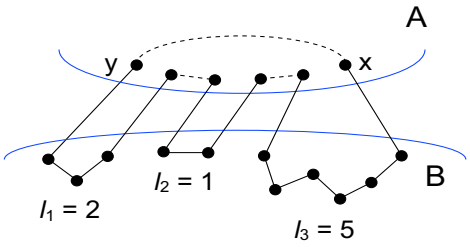

Figura 4.7: Subcaminhos maximais disjuntos em C.

Observação 4.14 [12] *Existe um buraco C com pelo menos dois vértices consecutivos em B se e somente se existir uma aresta (x, y), $x \in A$, $y \in B$ e um caminho entre um vértice em $(N(y) - N(x)) \cap B$ e um vértice em $N(x) - N(y)$ que evita quaisquer outros vértices em $N(x) \cup N(y)$.*

Para detectarmos se tal caminho existe, devemos: deletar $N(x) \cap N(y)$ e $(N(y) - N(x)) \cap A$ de G; encontrar as componentes conexas em G induzidas nos demais vértices; e checar se existe um vértice em $(N(y) - N(x)) \cap B$ e um vértice em $N(x) - N(y)$ na mesma componente conexa. Este processo necessita de tempo de $O(m)$ por aresta, podendo ser implementado de modo que, se o caminho existir, então o processo dará como saída um buraco através de (x, y) tal que o outro vizinho de y também está em B.

O número de vértices adicionados ao conjunto A depois da detecção de cada ciclo por P_2 é de no máximo duas vezes o aumento de cc. Como cc nunca é maior que k, o número total de vértices em A quando P_2 termina permanece $O(k)$.

(P_3) *Adicionando arestas essenciais em G_A:*

Para cada par de vértices não adjacentes $y, z \in V$, seja $A_{y,z}$ o conjunto de todos os vértices x tais que y, x, z apareçam consecutivamente em algum buraco em G.

Lema 4.15 [12] *Se, para algum par $y, z \in A$, $(y, z) \notin E$, temos que $|A_{y,z}| > 2k$, então a aresta (y, z) está em todas as k-triangulações de G.*

Prova: Assumindo que (y, z) não esteja em uma k-triangulação $\overline{G} = (V, \overline{E})$ de G, então deve existir uma corda em $\overline{E} - E$ incidente a cada vértice em $A_{y,z}$. Como apenas dois vértices compartilham uma corda, temos $|\overline{E} - E| > k$, uma contradição. □

As arestas (y, z) que satisfazem o Lema 4.15 são ditas *essenciais*. Para uma tripla y, x, z tal que $(y, x) \in E$, $(x, z) \in E$ e $(y, z) \notin E$, podemos determinar se y, x, z aparece consecutivamente em algum buraco em tempo linear, uma vez que isto ocorre se e somente se y e z estiverem em uma mesma componente conexa após a deleção de $N(x) - \{y, z\}$ de G.

O procedimento P_3 primeiro calcula os conjuntos $A_{y,z}$ para todo par $y, z \in A$, $(y, z) \notin E$. Depois, para cada par $y, z \in A$ tal que $|A_{y,z}| > 2k$, a aresta (y, z) é adicionada a G'. Finalmente, todos os vértices em cada conjunto $A_{y,z}$ tal que $|A_{y,z}| \leq 2k$ são adicionados a A.

As complexidades de tempo de P_2 e P_3 são estabelecidas a seguir:

Lema 4.16 [12] *(1) A execução de P_2 leva um tempo de $O(knm)$. (2) A execução de P_3 leva um tempo de $O(k^2nm)$.*

Prova:
(1) Para cada aresta (x, y), $x \in A$, $y \in B$, é possível encontrar um buraco através de (x, y) com vértices consecutivos em B em tempo linear. O tamanho do conjunto A é sempre de $O(k)$; assim, o número total de arestas incidentes com vértices em A é sempre de $O(kn)$. Para cada uma dessas arestas, poderá ser necessário executar o teste acima uma vez, gerando uma complexidade total de tempo de $O(knm)$.
(2) Como o tamanho de A quando P_3 começa sua execução é de $O(k)$, o número de triplas y, x, z tais que $(y, x), (z, x) \in E$, $(y, z) \notin E$, $y, z \in A$, é de $O(k^2n)$; dado por k vértices combinados dois a dois vezes $O(n)$ vértices restantes. Para cada tripla, é necessário verificar se existe um caminho entre y e z após a deleção de $N(x) - \{y, z\}$ de G. Como mencionado anteriormente, isto pode ser feito identificando-se as componentes conexas de G nos vértices restantes e então checando se y e z estão na mesma

componente conexa, o que requer um gasto de tempo linear. Assim, a complexidade total de tempo de P_3 é de $O(k^2nm)$. □

A complexidade total do procedimento de particionamento é dominada pela complexidade de P_3. Antes de sua chamada, o tamanho do conjunto A é $O(k)$. O procedimento P_3 pode adicionar $O(k)$ vértices ao conjunto A para cada par de vértices em A anterior à sua execução. Então, o conjunto A possuirá $O(k^3)$ vértices após o procedimento de particionamento.

Sejam E_S o conjunto de arestas essenciais detectadas por P_3 e $G' = (V, E \cup E_S)$. Denotemos por A_2, B_2 a partição do conjunto de vértices antes da execução de P_3 e por A, B a partição final.

Para estabelecermos a corretude do esquema de particionamento e a completude do algoritmo, precisamos do seguinte lema:

Lema 4.17 [12] *Seja $G = (V, E)$ um grafo e $v \in V$. Seja F um conjunto de arestas entre vértices de G tal que:*
i) Cada aresta $e \in F$ é uma corda de um buraco C_e de G.
ii) $F \cap E = \emptyset$.
iii) v não é extremidade de nenhuma aresta $e \in F$.
Denotemos por G^+ o grafo obtido de G pela adição das arestas de F. Se existir um buraco C em G^+ com v_1, v, v_2 ocorrendo consecutivamente em C, então, temos 2 possibilidades:
1) Existe um buraco em G no qual v_1, v, v_2 ocorrem consecutivamente, ou
2) Existe um buraco $D_e = v, x_1, \ldots, x_t, v$ em G tal que o caminho $p = x_1, \ldots, x_t$ é parte de um ciclo C_e para algum $e \in F$, e p contém uma das extremidades de e.

Prova: Seja C um buraco em G^+ no qual v_1, v, v_2 ocorrem consecutivamente. Considerando as arestas que compõem C, temos 2 casos possíveis:

Caso 1) Todas as arestas pertencem a $V(G)$. Logo, o próprio C é o buraco em G no qual v_1, v, v_2 ocorrem consecutivamente. Como exemplo, no grafo G^+ da Figura 4.8, temos $C = v_1, v, v_2, d, v_1$.

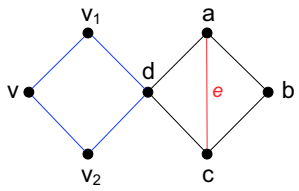

Figura 4.8: Exemplo de um ciclo para o Caso 1.

Caso 2) Existe pelo menos uma aresta de C que não pertencia a G, pertencendo, portanto, a F. Sejam $e_1, \ldots, e_k$ tais arestas. Como cada aresta e_i é uma corda de algum buraco C_{e_i} em G, podemos ilustrar C como na Figura 4.9.

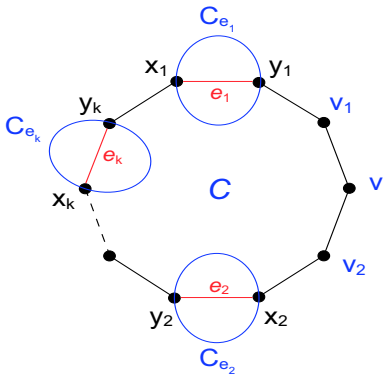

Figura 4.9: Identificação das arestas $e_1, \ldots, e_k$ em C pelo Caso 2.

Para uma aresta $e = (x, y) \in F$, sejam P_e^1 e P_e^2 os dois caminhos em C_e entre x e y, com x e y removidos de cada caminho. Como C_e é um buraco, para toda aresta e tal que $v \in C_e$, v não é adjacente a nenhum vértice em P_e^1 ou em P_e^2. Obviamente, v não pertence a nenhum buraco $C_{e_i}, 1 \leq i \leq k$, pois $v \in C$ e não pode ser extremo de nenhuma aresta $e \in F$.

Caso 2.1) Para toda aresta $e \in F$ tal que $v \notin C_e$, existe um caminho $P_e \in \{P_e^1, P_e^2\}$ tal que v não é adjacente em G a nenhum vértice em P_e. Consideremos o ciclo C. Substituindo cada aresta $e_i \in F$ ao longo de C por $P_{e_i}, 1 \leq i \leq k$, obtemos um ciclo C' em G (não necessariamente sem corda ou simples) com a propriedade de que v não é adjacente a nenhum vértice em $C' - \{v_1, v_2\}$.

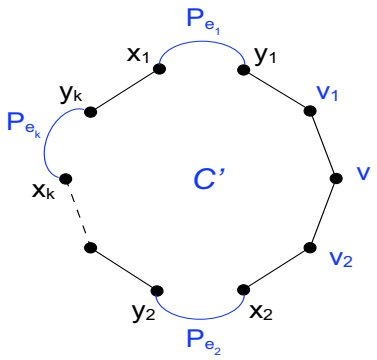

Figura 4.10: Ciclo C' identificado em G pelo Caso 2.1.

Como as arestas em F não são incidentes a v, as arestas (v, v_1) e (v, v_2) existem em G. Como C não possui cordas, $(v_1, v_2) \notin E$. Assim, as possíveis cordas de C' em G podem ser: 1) com um extremo em P_{e_i} e outro em P_{e_j} $(1 \leq i, j \leq k,\ i \neq j)$, ou 2) com um extremo em P_{e_i} e outro em $u \in C$, tal que $u \notin P_{e_i}$. Podemos, portanto, percorrer C', partindo de v e usando tais cordas como "atalhos" para voltar a v, gerando, dessa forma, um ciclo sem cordas C'', que é um buraco em G no qual v_1, v, v_2 ocorrem consecutivamente. A Figura 4.11 ilustra um exemplo deste processo.

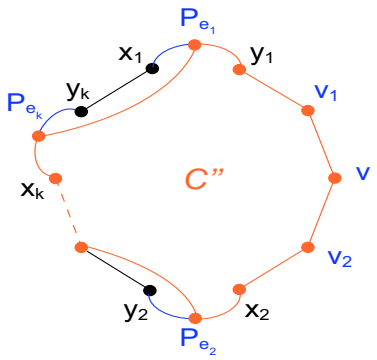

Figura 4.11: Buraco C'' identificado a partir de C'.

Caso 2.2: Para alguma aresta $e = (x, y) \in F$, v é adjacente a um vértice $u_1 \in P_e^1$ e a um vértice $u_2 \in P_e^2$.

Como C é um buraco em G^+, v necessariamente é não adjacente a x ou a y em G. Utilizando o exemplo da Figura 4.12, poderíamos ter v adjacente

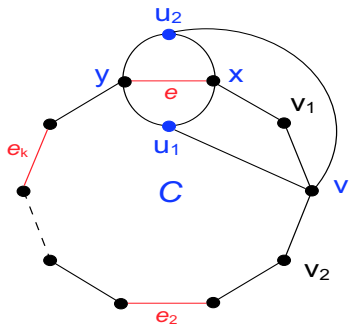

Figura 4.12: Vértice v adjacente a u_1 e u_2 pelo Caso 2.2.

a x, caso $v_1 = x$. Sem perda de generalidade, podemos assumir que v é não adjacente a x, e que u_1 e u_2 são os vértices mais próximos de x dentre todos os vértices em P_e^1 e P_e^2, respectivamente, que são adjacentes a v. D_e é um buraco em G consistindo do vértice v e de todos os vértices do caminho entre u_1 e u_2 através de x em C_e, como ilustrado na Figura 4.13.

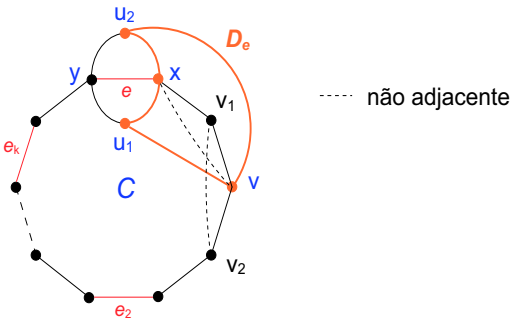

Figura 4.13: Buraco em G através de x em C_e.

□

A corretude do esquema de particionamento é obtida pelo seguinte teorema:

Teorema 4.18 [12] *Quando o procedimento de particionamento termina, o grafo G' não contém ciclos sem corda com vértices no conjunto B.*

Prova: A prova é dada por contradição. Suponhamos que exista um buraco $C \in G'$ tal que $C \cap B \neq \emptyset$, e seja v um vértice em $C \cap B$. Denotemos por v_1 e

v_2 os dois vizinhos de v em C. O ciclo C contém necessariamente pelo menos uma aresta essencial, pois, caso contrário, C existiria em G e v teria sido movido para A, ou a aresta (v_1, v_2) teria sido adicionada como uma aresta essencial. Seja F o conjunto de arestas essenciais em C. Pela definição de aresta essencial, para cada aresta $e = (x, y) \in F$, existe um buraco C_e em G no qual e é uma corda. Além do mais, se P_e^1 e P_e^2 são os dois caminhos conectando x e y em C_e, então P_e^1 ou P_e^2 consiste de um único vértice $z_e \in B_2$. Como v está em B, nenhuma aresta essencial é incidente a ele. Aplicando o Lema 4.17, temos que uma das seguintes possibilidades ocorre:

(1) Existe em G um buraco no qual v_1, v, v_2 ocorrem consecutivamente. Assim, ou v deveria ter sido movido para A ou a aresta (v_1, v_2) deveria ter sido adicionada como essencial, caracterizando uma contradição.

(2) Existe um buraco D_e em G no qual v e z_e ocorrem consecutivamente para algum $e \in F$. Como v e z_e estão em B_2, ambos deveriam ter sido movidos para A por P_2, e obtemos novamente uma contradição. □

Triangularizando o grafo

Uma vez que o grafo já tenha sido particionado, é suficiente então procurar por $(k - a)$-triangulações em um grafo menor com o conjunto de vértices A, onde a é o número de arestas essenciais adicionadas durante o algoritmo de particionamento. Este é o conteúdo do próximo teorema, cuja prova necessita do Teorema 4.5:

Teorema 4.19 [12] *Seja A, B uma partição do conjunto de vértices V de um grafo $G = (V, E)$, tal que os vértices de todo buraco em G estão contidos em A. Um conjunto de arestas F é uma triangulação minimal de G se e somente se F for uma triangulação minimal de G_A.*

Prova: Seja F uma triangulação minimal de G_A. É necessário provar que $G' = (V, E \cup F)$ é cordal. Por contradição, seja G' não cordal e seja C um buraco em G'. Como G' induzido em A é cordal, $C \cap B \neq \emptyset$. Por hipótese, G não contém buracos com vértices em B. Conseqüentemente, C não pode existir em G, e, assim, ele contém pelo menos uma aresta de F e $|C \cap A| \geq 2$. Seja v um vértice em $C \cap B$. Pelo Teorema 4.5, cada aresta $e \in F$ é uma

corda de um buraco C_e de G, cujos vértices estão em A. Como F é uma triangulação minimal de G_A, v não é extremidade de nenhuma aresta de F. Usando o Lema 4.17, conclui-se que necessariamente existe um buraco que contenha v em G, o que contradiz a hipótese do teorema.

Para provar a outra direção, seja F uma triangulação minimal de G. Seja $F' \subseteq F$ uma triangulação minimal de G_A. Pela primeira parte da prova, F' também triangula G. Como F é minimal, conclui-se que $F' = F$. □

Tempo de execução total

O passo final do algoritmo procura por $(k-a)$-triangulações no conjunto de vértices A. Tais triangulações podem ser encontradas usando o algoritmo descrito na seção 4.4.1. Como $|A| = \mathrm{O}(k^3)$, o tempo de execução deste passo é de $\mathrm{O}(k^6 2^{4k})$. O tempo total dos três passos do procedimento de particionamento é de $\mathrm{O}(k^2 nm)$, gerando um tempo total para o algoritmo de $\mathrm{O}(k^2 nm + k^6 2^{4k})$.

Capítulo 5

Algoritmos por parâmetro fixo para obtenção de grafos aglomerados

5.1 Introdução

Um grafo G é dito ser um *grafo aglomerado* (*cluster*) se cada uma de suas componentes conexas for uma clique.

Dado um grafo $G = (V, E)$ de entrada, o problema de aglomeração por modificação de arestas (PA-M) consiste em tornar G uma união de cliques disjuntas através do menor número de alterações (inclusões e/ou exclusões) no conjunto de arestas de G. Como caso especial, temos o problema de aglomeração por remoção de arestas (PA-R), no qual o grafo G deve ser transformado apenas através da deleção de arestas. Tais problemas são NP-completos.

Em suas versões parametrizadas – PA-M(k) e PA-R(k), temos como entrada um grafo G e um inteiro não negativo k. Assim, o PA-M(k) consiste em transformar G em uma união de cliques disjuntas executando no máximo k alterações (adição ou remoção) no conjunto E, enquanto que, no PA-R(k), podemos ter no máximo k remoções de arestas. Neste capítulo, serão apresentados dois algoritmos para tais parametrizações, propostos por J. Gramm et al. em [11], cujas complexidades de tempo são de $O(2.27^k + n^3)$ e $O(1.77^k + n^3)$, respectivamente. Em ambos, primeiramente obtém-se um

núcleo para o problema. Em seguida, o método de busca em árvore é aplicado ao núcleo obtido.

5.2 Preliminares e noções básicas

Os algoritmos de busca em árvore apresentados a seguir trabalham de forma recursiva, onde o número de chamadas recursivas é dado pelo número de nós da árvore correspondente. Este número é formado por recorrências lineares e homogêneas, com coeficientes constantes. Segundo [11], sua solução assintótica é determinada pelas raízes do polinômio característico. Se o algoritmo resolve um problema de tamanho s e chama a si próprio recursivamente para problemas de tamanhos $s - d_1$, ..., $s - d_i$, então, $(d_1, \ldots, d_i)$ é dito ser o *vetor de ramificação* da recursão. Ele corresponde à recorrência

$$t_s = t_{s-d_1} + \ldots + t_{s-d_i}$$

onde t_s denota o número de folhas na árvore de busca que resolve uma instância de tamanho s e $t_j = 1$ para $0 \leq j < d$, com $d = \max(d_1, \ldots, d_i)$. Esta recorrência corresponde ao *polinômio característico*

$$z^d = z^{d-d_1} + \ldots + z^{d-d_i}.$$

Se α é uma raiz do polinômio característico que tem valor absoluto máximo e é positivo, então t_s é α^s até um fator polinomial. α é chamado *número de ramificação*, que corresponde ao vetor de ramificação $(d_1, \ldots, d_i)$. Além disso, se α é uma raiz simples, então $t_s = \mathrm{O}(\alpha^s)$. Todos os números de ramificações que aparecerão neste capítulo são raízes simples.

Portanto, o tamanho da árvore de busca é de $\mathrm{O}(\alpha^k)$, onde k é o parâmetro e α é o maior número de ramificação que irá ocorrer. No PA-M(k), por exemplo, será mostrado que este número é aproximadamente 2.27, pertencendo ao vetor de ramificação (1, 2, 2, 3, 3).

Nos algoritmos apresentados a seguir, será usada uma tabela T para armazenar informações sobre as arestas do grafo, tal que T tem uma entrada para cada par de vértices $u, v \in V$, que pode ser vazia ou ter um dos seguintes valores:

"permanente": Neste caso, a aresta $(u, v) \in E$ e o algoritmo não pode mais removê-la; ou

"proibida": Neste caso, a aresta $(u, v) \notin E$ e o algoritmo não pode mais adicioná-la.

Devemos notar que, sempre que os algoritmos removem uma aresta (u, v) do conjunto E, $T[u, v]$ é alterado para proibida, uma vez que não faz sentido reintroduzir arestas previamente removidas. Da mesma forma, sempre que os algoritmos adicionam uma aresta (u, v) a E, $T[u, v]$ é alterado para permanente. Daqui por diante, quando tivermos a adição ou remoção de arestas, assume-se que estes ajustes serão feitos, mesmo que não mencionados explicitamente.

5.3 Núcleo do PA-M(k)

Uma *regra de redução* substitui, em tempo polinomial, uma dada instância (G, k) para o PA-M(k) por uma instância mais simples (G', k'), tal que (G, k) é uma instância com resposta positiva se e somente se (G', k') também o for. Assim sendo, G pode ser transformado em cliques disjuntas pela deleção/adição de no máximo k arestas se e somente se G' puder ser transformado em cliques disjuntas pela deleção/adição de no máximo k' arestas. Uma instância para a qual nenhum dos conjuntos de regras de redução dados se aplica é dita *reduzida* em relação a essas regras. Um problema parametrizado tal como PA-M(k) (sendo k o parâmetro) possui um *núcleo* se, após a aplicação de regras de redução, a instância reduzida resultante tiver tamanho $f(k)$, sendo f uma função que depende somente de k.

Para cada uma das regras de redução para o PA-M(k) apresentadas a seguir, sua corretude será discutida, bem como o tempo de execução necessário para tal regra. Através de seu uso, será mostrado que o núcleo do problema consiste de no máximo $2(k^2 + k)$ vértices e $(k^3 + k^2)$ arestas.

Embora as regras de redução também possam *adicionar* arestas ao grafo, podemos considerar a instância resultante como *simplificada*. Isto se deve ao fato de que, para cada aresta adicionada, o parâmetro tem seu valor decrementado de um. Para cada regra, é implicitamente assumido que, quando

uma aresta é adicionada ou removida, o valor do parâmetro é decrementado de um.

Na formulação das regras, é usada a seguinte terminologia. Dados um grafo $G = (V, E)$ e um par de vértices $v_i, v_j \in V$, um vértice $z \in V$ é *vizinho comum* a v_i e v_j se $(z, v_i) \in E$ e $(z, v_j) \in E$. Similarmente, $z \in V$ é *vizinho não comum* a v_i e v_j, se $z \neq v_i$ e $z \neq v_j$ e, ou $(z, v_i) \in E$, ou $(z, v_j) \in E$, mas não ambas.

Regra 1 Para todo par de vértices $u, v \in V$:

1. Se u e v têm mais de k vizinhos comuns, então a aresta (u, v) tem que pertencer a E, resultando em $T[u, v] :=$ permanente. Se $(u, v) \notin E$, então a adicionamos.

2. Se u e v têm mais de k vizinhos não comuns, então a aresta (u, v) não pode pertencer a E, resultando em $T[u, v] :=$ proibida. Se $(u, v) \in E$, então a removemos.

3. Se u e v têm mais de k vizinhos comuns e mais de k vizinhos não comuns, então a instância dada não tem solução.

Lema 5.1 [11] *A Regra 1 é correta.*

Prova:
Caso 1: Os vértices u e v têm mais de k vizinhos comuns. Se a aresta (u, v) fosse excluída de E, então, para cada vizinho comum z de u e v, pelo menos uma das arestas $(u, z), (v, z)$ também teria de ser removida. Isto, entretanto, necessitaria de mais de $k + 1$ remoções de arestas, contrariando o máximo de k modificações de arestas permitidas.
Caso 2: Os vértices u e v têm mais de k vizinhos não comuns. Se a aresta (u, v) fosse incluída em E, então, para cada vizinho não comum z de u e v, seria necessário editar uma das arestas (u, z) e (v, z). Sem perda de generalidade, seja z vizinho a u e não vizinho a v. Então, seria necessário remover a aresta (u, z) de E ou adicionar a aresta (v, z) a E. Como existem pelo menos $k + 1$ vizinhos não comuns, seriam necessárias pelo menos $k + 1$

modificações no conjunto de arestas.
Caso 3: Os vértices u e v têm mais de k vizinhos comuns e mais de k vizinhos não comuns. A partir das provas dos casos 1 e 2, temos que seriam necessárias mais de k modificações de arestas, tanto para incluir a aresta (u, v) a E quanto para excluí-la de E. □

Podemos notar que a Regra 1 se aplica a qualquer par de vértices $\{u, v\}$ cujo número de vértices que são vizinhos a u ou v (ou a ambos) é maior que $2k$.

Regra 2 Para toda tripla de vértices $u, v, w \in V$:

1. Se $T[u, v]$ = permanente e $T[u, w]$ = permanente, então a aresta (v, w), caso ainda não exista, tem que ser adicionada a E e $T[v, w] :=$ permanente.

2. Se $T[u, v]$ = permanente e $T[u, w]$ = proibida, então a aresta (v, w), caso exista, tem que ser removida de E e $T[v, w] :=$ proibida.

A corretude da Regra 2 é óbvia. A análise do tempo de execução da aplicação das Regras 1 e 2, que se dá de forma intercalada, é feita a seguir.

Lema 5.2 [11] *Um grafo pode ser transformado em um tempo de $O(n^3)$ em um grafo reduzido em relação às Regras 1 e 2.*

Prova: A prova se dá pela apresentação de um algoritmo que reduz um dado grafo $G = (V, E)$ em um grafo $G' = (V', E')$ reduzido em relação às Regras 1 e 2 em um tempo de $O(n^3)$. O algoritmo processa as Regras 1.1 e 1.2 e, com um pequeno esforço adicional, também as Regras 1.3 e 2.
Algoritmo para prova do Lema 5.2:

Estruturas de Dados

- Matriz de adjacências de G.

- Dois arranjos $((n-1) \times n)/2$, chamados C e N onde, para cada par de vértices $\{v_i, v_j\}$, com $i < j$, $C[i, j]$ contém o número de vizinhos comuns a v_i e v_j e $N[i, j]$, o número de vizinhos não comuns.

- Listas encadeadas para armazenar os pares de vértices que são candidatos a serem inseridos no conjunto de arestas E ou a serem retirados do conjunto. Mais precisamente, são mantidas as listas $L_{c,0}, L_{c,1}, \ldots, L_{c,k}$, e L_p, onde $L_{c,r}$, $0 \leq r \leq k$, contém os pares de vértices que têm exatamente r vizinhos comuns. A lista L_p contém os pares de vértices que estão marcados para serem configurados como permanentes pela Regra 1.1 ou pela Regra 2; por exemplo, por terem mais de k vizinhos comuns. Similarmente, são mantidas as listas $L_{n,0}, L_{n,1}, \ldots, L_{n,k}$, e L_f, onde $L_{n,s}$, $0 \leq s \leq k$, contém os pares de vértices que têm s vizinhos não comuns. A lista L_f contém os pares de vértices que estão marcados para que as arestas entre eles sejam proibidas pela Regra 1.2 ou pela Regra 2.

- Dois arranjos $((n-1) \times n)/2$, chamados P_c e P_n. Um par de vértices $\{v_i, v_j\}$, $i < j$, está contido em no máximo uma lista dentre $L_{c,0}, L_{c,1}, \ldots, L_{c,k}$. Se $\{v_i, v_j\}$ estiver contido em uma destas listas, então a entrada do arranjo $P_c[i,j]$ contém um ponteiro para esta entrada da lista. Se $\{v_i, v_j\}$ não estiver contido em nenhuma destas listas, então $P_c[i,j]$ contém um ponteiro nulo. Analogamente, $P_n[i,j]$ contém ou um ponteiro nulo ou um ponteiro para uma entrada de $\{v_i, v_j\}$ em uma lista dentre $L_{n,0}, L_{n,1}, \ldots, L_{n,k}$.

Inicialização. Assume-se que a matriz de adjacências de G é dada. Para cada par de vértices $v_i, v_j \in V$, $i < j$, $C[i,j]$ e $N[i,j]$ são inicializados com o número de vizinhos comuns e não comuns, respectivamente, na matriz de adjacências. Baseado nesses números, o par de vértices $\{v_i, v_j\}$ é adicionado à lista apropriada. Se v_i e v_j têm r vizinhos comuns, $0 \leq r \leq k$, então $\{v_i, v_j\}$ é adicionado à lista $L_{c,r}$. Se v_i e v_j têm mais de k vizinhos comuns, então $\{v_i, v_j\}$ é adicionado a L_p. Analogamente, $\{v_i, v_j\}$ é adicionado à lista $L_{n,s}$ caso v_i e v_j tenham s vizinhos não comuns, $0 \leq s \leq k$, ou adicionado a L_f caso tenham mais de k vizinhos não comuns. Se um par de vértices é adicionado a uma das listas $L_{c,0}, L_{c,1}, \ldots, L_{c,k}$, então um ponteiro para esta entrada é armazenado para este par de vértices em $P_c[i,j]$ (analogamente em $P_n[i,j]$ se o par de vértices for adicionado em uma das

listas $L_{n,0}, L_{n,1}, \ldots, L_{n,k}$). Se, nesta etapa ou no seguinte algoritmo, um par de vértices $\{v_i, v_j\}$ satisfizer $C[i,j] > k$ e $N[i,j] > k$, então o algoritmo termina, reportando que esta instância não tem solução devido à Regra 1.3.

Para ilustrarmos a inicialização, consideremos como entrada o grafo G da Figura 5.1 e $k = 2$.

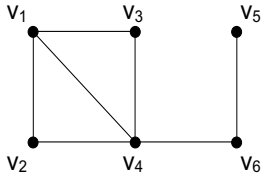

Figura 5.1: Grafo não aglomerado.

A partir da matriz de adjacências, dada como entrada, as estruturas de dados são inicializadas como a seguir. A tabela T será omitida, por ainda conter todas as suas entradas vazias.

C :

	v_1	v_2	v_3	v_4	v_5	v_6
v_1		1	1	2	0	1
v_2			2	1	0	1
v_3				1	0	1
v_4					1	0
v_5						0

N :

	v_1	v_2	v_3	v_4	v_5	v_6
v_1		1	1	1	4	3
v_2			0	2	3	2
v_3				2	3	2
v_4					3	4
v_5						1

$L_{c,0}$:

$\boxed{v_1,\ v_5} \rightarrow \boxed{v_2,\ v_5} \rightarrow \boxed{v_3,\ v_5} \rightarrow \boxed{v_4,\ v_6} \rightarrow \boxed{v_5,\ v_6}$

$L_{c,1}$:

$\boxed{v_1,\ v_2} \rightarrow \boxed{v_1,\ v_3} \rightarrow \boxed{v_1,\ v_6} \rightarrow \boxed{v_2,\ v_4} \rightarrow \boxed{v_2,\ v_6} \rightarrow \boxed{v_3,\ v_4} \rightarrow$

$\boxed{v_3,\ v_6} \rightarrow \boxed{v_4,\ v_5}$

$L_{c,2}$:

$\boxed{v_1,\ v_4} \rightarrow \boxed{v_2,\ v_3}$

$L_p : \emptyset$

$L_{n,0}$:

$\boxed{v_2,\ v_3}$

$L_{n,1}$:

$\boxed{v_1,\ v_2} \rightarrow \boxed{v_1,\ v_3} \rightarrow \boxed{v_1,\ v_4} \rightarrow \boxed{v_5,\ v_6}$

$L_{n,2}$:

$\boxed{v_2,\ v_4} \rightarrow \boxed{v_2,\ v_6} \rightarrow \boxed{v_3,\ v_4} \rightarrow \boxed{v_3,\ v_6}$

L_f :

$\boxed{v_1,\ v_5} \rightarrow \boxed{v_1,\ v_6} \rightarrow \boxed{v_2,\ v_5} \rightarrow \boxed{v_3,\ v_5} \rightarrow \boxed{v_4,\ v_5} \rightarrow \boxed{v_4,\ v_6}$

P_c :

	1	2	3	4	5	6
1		$L_{c,1}(1)$	$L_{c,1}(2)$	$L_{c,2}(1)$	$L_{c,0}(1)$	$L_{c,1}(3)$
2			$L_{c,2}(2)$	$L_{c,1}(4)$	$L_{c,0}(2)$	$L_{c,1}(5)$
3				$L_{c,1}(6)$	$L_{c,0}(3)$	$L_{c,1}(7)$
4					$L_{c,1}(8)$	$L_{c,0}(4)$
5						$L_{c,0}(5)$

P_n :

	1	2	3	4	5	6
1		$L_{n,1}(1)$	$L_{n,1}(2)$	$L_{n,1}(3)$	$\emptyset$	$\emptyset$
2			$L_{n,0}(1)$	$L_{n,2}(1)$	$\emptyset$	$L_{n,2}(2)$
3				$L_{n,2}(3)$	$\emptyset$	$L_{n,2}(4)$
4					$\emptyset$	$\emptyset$
5						$L_{n,1}(4)$

Nos arranjos P_c e P_n, a notação $L_{c,i}(j)$ (ou $L_{n,i}(j)$) indica um ponteiro para o j-ésimo elemento da lista $L_{c,i}$ (ou $L_{n,i}$), $0 \leq i \leq k$.

Algoritmo. A seguir será descrita uma iteração do algoritmo. O algoritmo termina quando as listas L_p e L_f estiverem vazias ou quando k assumir valor negativo. Se pelo menos uma delas não estiver vazia, o par de vértices contido em L_p ou L_f está aguardando ser processado. Cada iteração do algoritmo processa um destes pares de vértices. Será descrito como processar um par de vértices $\{v_i, v_j\}$, $i < j$, proveniente de L_p. O processamento de um par de L_f é realizado de forma análoga, conforme demonstrado em exemplo.

Para um par de vértices $\{v_i, v_j\}$ de L_p, v_i e v_j são marcados para serem conectados por uma aresta permanente devido à Regra 1 ou à Regra 2. Caso a aresta (v_i, v_j) ainda não esteja no conjunto de arestas, deve-se acrescentá-la e decrementar o parâmetro k em uma unidade, configurando $T[i, j] :=$ permanente.

Quando se adiciona uma nova aresta ao conjunto de arestas,

(a) é necessário atualizar os contadores de vizinhos comuns e não comuns. Caso algum contador seja alterado, devem-se atualizar também as listas.

Quando se altera o valor de uma entrada em T,

(b) precisa-se testar se este novo valor torna necessária uma aplicação da Regra 2.

Considerando (a), assume-se que foi adicionada uma nova aresta (v_i, v_j). Seja k o parâmetro ainda com seu valor antigo, ou seja, não decrementado. Para atualizar os contadores e testar se as listas têm que ser atualizadas, consideremos somente os vértices v_l, $v_l \neq v_i$ e $v_l \neq v_j$, tais que, para $\{v_i, v_l\}$ e $\{v_j, v_l\}$, os contadores de vizinhos comuns e não comuns possam mudar. Como uma nova aresta foi adicionada, devemos considerar as seguintes situações (sem perda de generalidade, assume-se que $i < j < l$):

- O vértice v_l é um vizinho comum a v_i e v_j. Então, configura-se $N[i,l] := N[i,l]-1$, $C[i,l] := C[i,l]+1$, $N[j,l] := N[j,l]-1$, e $C[j,l] := C[j,l]+1$. A título de exemplo, se acrescentássemos a aresta (v_3, v_6) ao grafo da Figura 5.1 ($i = 3$ e $j = 6$), para o vértice $v_l = v_4$, passaríamos a ter: $N[3,4] = 1$, $C[3,4] = 2$, $N[4,6] = 3$ e $C[4,6] = 1$.

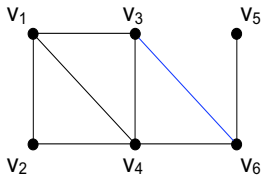

Figura 5.2: Acréscimo da aresta (v_3, v_6).

- O vértice v_l é vizinho de apenas um vértice dentre o par $\{v_i, v_j\}$. Sem perda de generalidade, seja v_l vizinho de v_j e não vizinho de v_i. Então, configura-se $N[i,l] := N[i,l] - 1$, $C[i,l] := C[i,l] + 1$, e $N[j,l] := N[j,l] + 1$. Seguindo o exemplo anterior, se considerarmos $v_l = v_5$, obtemos como resultado das atualizações: $N[3,5] = 2$, $C[3,5] = 1$ e $N[5,6] = 2$.

- O vértice v_l não é vizinho de v_i nem de v_j. Então, configura-se $N[i,l] := N[i,l] + 1$ e $N[j,l] := N[j,l] + 1$. Seguindo novamente o exemplo e considerando agora $v_l = v_2$, obtemos $N[2,3] = 1$ e $N[2,6] = 3$.

Se o valor de uma entrada em C mudar, atualizamos as listas $L_{c,r}$, $0 \leq r \leq k$, e L_p. Se o valor de uma entrada em N mudar, atualizamos as listas $L_{n,s}, 0 \leq s \leq k$, e L_f. A atualização dessas listas é descrita através do exemplo do incremento de $C[i,l]$. Se $C[i,l]$ for incrementado para um valor de no máximo $k+1$ (pois, caso contrário, o par de vértices $\{v_i, v_l\}$ já estaria em L_p), e $\{v_i, v_l\}$ estiver contido uma das listas $L_{c,r}$, $0 \leq r \leq k$, então $\{v_i, v_l\}$ está contido em $L_{c,C[i,l]-1}$ (em relação ao novo valor de $C[i,l]$). Removemos $\{v_i, v_l\}$ de $L_{c,C[i,l]-1}$, usando o ponteiro armazenado em $P_c[i,l]$. Se o novo valor de $C[i,l]$ satisfizer $C[i,l] = k+1$, movemos a entrada para o fim da lista L_p; caso contrário, movemos a entrada para o fim da lista $L_{c,C[i,l]}$.

Então, removemos $\{v_i, v_l\}$ de L_p e removemos o ponteiro armazenado em $P_c[i, l]$, uma vez que não será mais necessário.

Com os contadores e listas atualizados, juntamos a lista $L_{c,k}$ (ainda em relação ao valor antigo de k) ao final da lista L_p, já que k será decrementado de uma unidade e os pares de vértices contidos em $L_{c,k}$ passarão a satisfazer a Regra 1.1. Da mesma forma, juntamos a lista $L_{n,k}$ ao final da lista L_f, devido à Regra 1.2. Então, atualizamos k, decrementando-o de 1.

Considerando (b), testamos se a aresta permanente (v_i, v_j) torna necessária uma aplicação da Regra 2, como a seguir. Novamente, consideremos cada vértice v_l, $v_l \neq v_i$ e $v_l \neq v_j$, e testamos se (v_i, v_l) é permanente mas não (v_j, v_l), ou vice-versa. Sem perda de generalidade, seja (v_i, v_l) a aresta permanente. Se ainda não estiver contido em L_p ou L_f, nós adicionamos o par de vértices não permanente $\{v_j, v_l\}$ à lista L_p.

Ambos os testes (a) e (b) podem ser feitos em um tempo de $O(n)$, usando-se a matriz de adjacências e os arranjos P_c e P_n. Isto completa a descrição da iteração que processa o par de vértices $\{v_i, v_j\}$. Quando este é processado, sua entrada correspondente é removida de L_p.

A iteração esboçada acima é repetida até que k assuma valor negativo ou L_p e L_f estejam vazias. Se o parâmetro k alcançar um valor negativo antes que L_p e L_f estejam vazias, então a instância dada não tem solução. Se L_p e L_f se tornarem vazias enquanto $k \geq 0$, não existe nenhum par de vértices restante para o qual a Regra 1 ou a Regra 2 se aplique. Logo, o grafo resultante é reduzido em relação às Regras 1 e 2.

Como exemplo, serão esboçadas as iterações da execução do algoritmo para o grafo da Figura 5.1.

1ª iteração: O par $\{v_1, v_5\}$, proveniente de L_f, é processado.

- Configura-se $T[1, 5] :=$ *proibida*. Como a aresta (v_1, v_5) já não existia em G, basta remover o par de L_f.
- Verifica-se que a Regra 2 não se aplica.

2^a *iteração:* O par $\{v_1, v_6\}$, proveniente de L_f, é processado.

- Configura-se $T[1,6] := \textit{proibida}$, bastando remover o par de L_f.
- Verifica-se que a Regra 2 não se aplica.

3^a *iteração:* O par $\{v_2, v_5\}$, proveniente de L_f, é processado.

- Configura-se $T[2,5] := \textit{proibida}$, removendo o par de L_f.
- A Regra 2 não se aplica.

4^a *iteração:* O par $\{v_3, v_5\}$, proveniente de L_f, é processado.

- Configura-se $T[3,5] := \textit{proibida}$, removendo o par de L_f.
- A Regra 2 não se aplica.

5^a *iteração:* O par $\{v_4, v_5\}$, proveniente de L_f, é processado.

- Configura-se $T[4,5] := \textit{proibida}$, removendo o par de L_f.
- A Regra 2 não se aplica.

6^a *iteração:* O par $\{v_4, v_6\}$, proveniente de L_f, é processado.

- Configura-se $T[4,6] := \textit{proibida}$, removendo o par de L_f. Como a aresta $(v_4, v_6) \in E$, esta é removida do grafo.
- Verifica-se que a Regra 2 não se aplica.

Devido à remoção de (v_4, v_6), os contadores de vizinhos comuns e não comuns devem ser atualizados. Para isso, consideremos as seguintes situações, similares às descritas no algoritmo (sendo $i < j < l$):

- O vértice v_l é vizinho comum a v_i e v_j. Então, configura-se $C[i,l] := C[i,l]-1$, $N[i,l] := N[i,l]+1$, $C[j,l] := C[j,l]-1$ e $N[j,l] := N[j,l]+1$.
  No exemplo, não há nenhum vértice que se encaixe neste caso.
- O vértice v_l não é vizinho de v_i nem de v_j. Então, configura-se $N[i,l] := N[i,l] - 1$ e $N[j,l] := N[j,l] - 1$.
  No exemplo, não há nenhum vértice que se encaixe neste caso.

- O vértice v_l é vizinho de apenas um vértice dentre $\{v_i, v_j\}$. Seja v_l vizinho de v_j mas não de v_i. Configura-se $C[i,l] := C[i,l] - 1$, $N[i,l] := N[i,l] + 1$ e $N[j,l] := N[j,l] - 1$.
  Nesta situação, encontram-se os vértices v_1, v_2, v_3 e v_5 do exemplo. O algoritmo realiza, então, as seguintes configurações:
  – por v_1: $C[1,6] := 0$, $N[1,6] := 4$ e $N[1,4] := 0$.
  – por v_2: $C[2,6] := 0$, $N[2,6] := 3$ e $N[2,4] := 1$.
  – por v_3: $C[3,6] := 0$, $N[3,6] := 3$ e $N[3,4] := 1$.
  – por v_5: $C[4,5] := 0$, $N[4,5] := 4$ e $N[5,6] := 0$.

Após cada alteração em C ou em N, as listas são atualizadas. Depois das alterações listadas acima, as estruturas de dados relevantes ao andamento do exemplo encontram-se da seguinte forma:

T :

	v_1	v_2	v_3	v_4	v_5	v_6
v_1		$\emptyset$	$\emptyset$	$\emptyset$	*proibida*	*proibida*
v_2			$\emptyset$	$\emptyset$	*proibida*	$\emptyset$
v_3				$\emptyset$	*proibida*	$\emptyset$
v_4					*proibida*	*proibida*
v_5						$\emptyset$

C :

	v_1	v_2	v_3	v_4	v_5	v_6
v_1		1	1	2	0	0
v_2			2	1	0	0
v_3				1	0	0
v_4					0	0
v_5						0

N :

	v_1	v_2	v_3	v_4	v_5	v_6
v_1		1	1	0	4	4
v_2			0	1	3	3
v_3				1	3	3
v_4					4	4
v_5						0

$L_{c,0}$:

$\boxed{v_1,\ v_5} \rightarrow \boxed{v_2,\ v_5} \rightarrow \boxed{v_3,\ v_5} \rightarrow \boxed{v_4,\ v_6} \rightarrow \boxed{v_5,\ v_6} \rightarrow \boxed{v_1,\ v_6} \rightarrow$

$\boxed{v_2,\ v_6} \rightarrow \boxed{v_3,\ v_6} \rightarrow \boxed{v_4,\ v_5}$

$L_{c,1}$:

$\boxed{v_1,\ v_2} \rightarrow \boxed{v_1,\ v_3} \rightarrow \boxed{v_2,\ v_4} \rightarrow \boxed{v_3,\ v_4}$

L_p :

$\boxed{v_1,\ v_4} \rightarrow \boxed{v_2,\ v_3}$

$L_{n,0}$:

$\boxed{v_2,\ v_3} \rightarrow \boxed{v_1,\ v_4} \rightarrow \boxed{v_5,\ v_6}$

$L_{n,1}$:

$\boxed{v_1,\ v_2} \rightarrow \boxed{v_1,\ v_3} \rightarrow \boxed{v_2,\ v_4} \rightarrow \boxed{v_3,\ v_4}$

L_f :

$\boxed{v_2,\ v_6} \rightarrow \boxed{v_3,\ v_6}$

Uma vez que as listas $L_{c,k}$ e L_p, assim como $L_{n,k}$ e L_f, já tenham sido unidas, podemos decrementar k, que assume valor 1.

7^a *iteração:* O par $\{v_2, v_6\}$, proveniente de L_f, é processado.

- Configura-se $T[2,6] := \textit{proibida}$, removendo o par de L_f.
- Verifica-se que a Regra 2 não se aplica.

8^a *iteração:* O par $\{v_3, v_6\}$, proveniente de L_f, é processado.

- Configura-se $T[3,6] := \textit{proibida}$, removendo o par de L_f.
- A Regra 2 não se aplica.

9^a *iteração:* O par $\{v_1, v_4\}$, proveniente de L_p, é processado.

- Configura-se $T[1,4] := \textit{permanente}$, removendo o par de L_p.
- Verifica-se que não há necessidade da Regra 2 ser aplicada.

10ª iteração: O par $\{v_2, v_3\}$, proveniente de L_p, é processado.

- Configura-se $T[2,3] := \textit{permanente}$, removendo o par de L_p.
- Verifica-se que não há necessidade da Regra 2 ser aplicada.

Como a aresta (v_2, v_3) ainda não pertence ao conjunto E, ela é acrescentada, tornando necessária a atualização de C, N e das listas. Após a atualização e a união das listas $L_{c,k}$ e L_p, bem como $L_{n,k}$ e L_f, decrementa-se k, que assume valor 0.

C :

	v_1	v_2	v_3	v_4	v_5	v_6
v_1		2	2	2	0	0
v_2			2	2	0	0
v_3				2	0	0
v_4					0	0
v_5						0

N :

	v_1	v_2	v_3	v_4	v_5	v_6
v_1		0	0	0	4	4
v_2			0	0	4	4
v_3				0	4	4
v_4					4	4
v_5						0

$L_{c,0}$:

$\boxed{v_1,\ v_5} \rightarrow \boxed{v_2,\ v_5} \rightarrow \boxed{v_3,\ v_5} \rightarrow \boxed{v_4,\ v_6} \rightarrow \boxed{v_5,\ v_6} \rightarrow \boxed{v_1,\ v_6} \rightarrow$
$\boxed{v_2,\ v_6} \rightarrow \boxed{v_3,\ v_6} \rightarrow \boxed{v_4,\ v_5}$

L_p :

$\boxed{v_1,\ v_2} \rightarrow \boxed{v_1,\ v_3} \rightarrow \boxed{v_2,\ v_4} \rightarrow \boxed{v_3,\ v_4}$

$L_{n,0}$:

$\boxed{v_2,\ v_3} \rightarrow \boxed{v_1,\ v_4} \rightarrow \boxed{v_5,\ v_6} \rightarrow \boxed{v_1,\ v_2} \rightarrow \boxed{v_1,\ v_3} \rightarrow \boxed{v_2,\ v_4} \rightarrow$
$\boxed{v_3,\ v_4}$

$L_f : \emptyset$

11ª iteração: O par $\{v_1, v_2\}$, proveniente de L_p, é processado.

- Configura-se $T[1,2] :=$ *permanente*, removendo o par de L_p.
- Pela Regra 2, configuram-se $T[1,3] :=$ *permanente* e $T[2,4] :=$ *permanente*.

12ª iteração: O par $\{v_1, v_3\}$, proveniente de L_p, é processado.

- Faz-se novamente $T[1,3] :=$ *permanente*, removendo o par de L_p.
- Pela Regra 2, configura-se $T[3,4] :=$ *permanente*.

13ª iteração: O par $\{v_2, v_4\}$, proveniente de L_p, é processado.

- Faz-se novamente $T[2,4] :=$ *permanente*, removendo o par de L_p.

14ª iteração: O par $\{v_3, v_4\}$, proveniente de L_p, é processado.

- Faz-se novamente $T[3,4] :=$ *permanente*, removendo o par de L_p.

Chegamos ao fim da execução do algoritmo com $k = 0$ e as listas L_p e L_f vazias. Logo, o grafo resultante da Figura 5.3 é reduzido em relação às Regras 1 e 2.

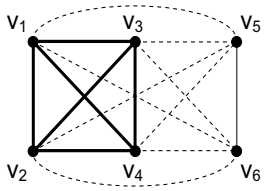

Figura 5.3: Grafo G' obtido a partir do grafo da Figura 5.1 pela aplicação do algoritmo. As linhas pontilhadas representam as arestas proibidas e as linhas em negrito, as arestas permanentes.

Tempo de execução. A inicialização das listas e arranjos, isto é, a contagem dos números de vizinhos comuns e não comuns para cada par de vértices pode ser feito em um tempo de $O(n^3)$: Para cada par de vértices $\{v_i, v_j\}$, consideremos todos os vértices v_l tais que $v_l \neq v_i$ e $v_l \neq v_j$. Uma entrada para $\{v_i, v_j\}$ é adicionada nas listas apropriadas em um tempo constante.

Uma iteração do algoritmo gasta no máximo um tempo de O(n), já que as entradas das listas podem ser acessadas e movidas em um tempo constante, usando os ponteiros armazenados nos arranjos P_c e P_n. Além disso, a iteração envolve somente um loop sobre todos os vértices de V.

Existem no máximo n^2 iterações (uma para cada par de vértices), já que cada par é processado no máximo uma vez, procedente de L_p ou de L_f. Concluindo, o tempo total para a redução de um dado grafo é de O(n^3).

Fim do algoritmo para prova do Lema 5.2 □

A regra a seguir completa o conjunto de regras de redução proposto nesta seção:

Regra 3 Remover todas as componentes conexas que sejam cliques do grafo.

A corretude da Regra 3 é óbvia. A computação das componentes conexas de um grafo, checando se estas são cliques, pode facilmente ser feito em um tempo linear:

Lema 5.3 [11] *A Regra 3 pode ser executada em um tempo de O(n).* □

Obviamente, para o núcleo do problema ser gerado, as Regras 1 e 3 seriam suficientes. A Regra 2 também é levada em conta por ser de fácil aplicabilidade, podendo ser considerada durante a execução da Regra 1. Assim, as Regras 1 e 3 constituem um pequeno conjunto de regras de redução fáceis e gerais, que levam a um núcleo do problema com O(k^2) vértices e O(k^3) arestas, sendo computável em um tempo de O(n^3).

O teorema a seguir mostra a corretude das regras, provando que reduzir um grafo em relação às Regras 1 e 3 conduz a um núcleo para o PA-M(k).

Teorema 5.4 [11] *O PA-M(k) possui um núcleo que contém no máximo* $2(k^2 + k)$ *vértices e no máximo* $k^3 + k^2$ *arestas, podendo ser encontrado em um tempo de O(*n^3*).*

Prova: Seja $G = (V, E)$ um grafo reduzido em relação às Regras 1 e 3. Sem perda de generalidade, seja G conexo. Caso contrário, cada componente conexa seria processada separadamente. Como a Regra 3 remove todas as cliques isoladas do grafo dado, G não é uma clique. Assim, é necessária ao menos uma modificação de aresta para transformar G em $G' = (V, E')$, que consista em cliques disjuntas. Seja k esse número mínimo, formado por k_a adições e k_d remoções de arestas. Pela hipótese de que G é reduzido em relação às Regras 1 e 3, será mostrado por contradição que $n \leq 2(k+1) \cdot k$ e que $m \leq 2\binom{k+1}{2}k$.

Seja $n > 2(k+1) \cdot k$. Temos, então, dois casos distintos: $k_a = 0$ ou $1 \leq k_a \leq k$.

Caso 1: $k_a = 0$. Temos k_d deleções de arestas, $1 \leq k_d = k$, para transformar G em G'. Seja $V_C \subset V$ o conjunto de vértices de uma clique de maior tamanho em G'. Os vértices em V_C também formam uma clique em G, já que $k_a = 0$. Como G é conexo, existe pelo menos um vértice $u \in V_C$ conectado a um vértice $v \notin V_C$. Então, ou v não está conectado a nenhum outro vértice $u' \in V_C$, com $u' \neq u$, ou existe um vértice $u' \in V_C$, com $u' \neq u$ tal que $(u', v) \in E$.

Caso 1.1: O vértice v não está conectado a nenhum outro vértice $u' \in V_C$, com $u' \neq u$, como exemplificado na Figura 5.4.

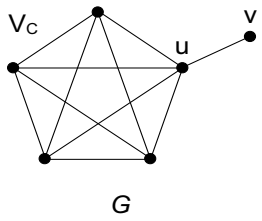

Figura 5.4: Vértice v vizinho somente a u em V_C.

Podemos limitar o tamanho da clique por $|V_C| \geq n/(k_d + 1)$: devido às k_d deleções de arestas, G é transformado em um grafo G' contendo no máximo $k_d + 1$ cliques. Conseqüentemente, uma clique de maior tamanho

em G' contém pelo menos $n/(k_d+1)$ vértices. Usando as hipóteses já citadas: $n > 2(k+1)\cdot k$ $^{(*)}$, $k_d \geq 1$ $^{(**)}$, e $k_d = k$ $^{(***)}$, obtemos

$$|V_C| \geq n/(k_d+1) \overset{(*)}{>} (2(k+1)\cdot k)/(k_d+1) \overset{(**)}{\geq} (2(k+1)\cdot k)/(2k_d) \overset{(***)}{=} k+1.$$

Logo, $|V_C| \geq k+2$ e u tem pelo menos $k+1$ vizinhos − todos os vértices $u' \in V_C$, com $u' \neq u$ − que não são vizinhos de v. Isto contradiz a hipótese de que G é reduzido em relação à Regra 1.

Caso 1.2: Existe um vértice $u' \in V_C$, com $u' \neq u$ tal que $(u', v) \in E$.

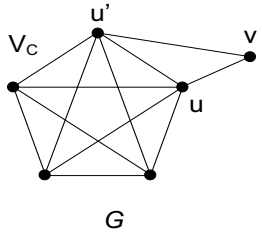

Figura 5.5: Vértice v vizinho a u e u' em V_C.

Podemos limitar o tamanho mínimo da clique por n/k_d: G é transformado em um grafo G' contendo no máximo k_d cliques. Conseqüentemente, uma clique de maior tamanho em G' contém pelo menos n/k_d vértices. Usando as hipóteses já citadas: $n > 2(k+1)\cdot k$ $^{(*)}$ e $k_d = k$ $^{(**)}$, obtemos

$$|V_C| \geq n/k_d \overset{(*)}{>} (2(k+1)\cdot k)/k_d \overset{(**)}{=} 2(k+1).$$

Logo, $|V_C| > 2k+2$ e no máximo k desses vértices estão conectados a v (pois $v \notin V_c$ e no máximo k arestas podem ser removidas). Então, u tem mais que $k+1$ vizinhos nesta clique que não são vizinhos de v, contradizendo a hipótese de que G é reduzido em relação à Regra 1.

Caso 2: $1 \leq k_a \leq k$. Como $k_a + k_d = k$, temos $k_d < k$. Novamente, seja $V_C \subset V$ o conjunto de vértices de uma clique de maior tamanho em G'. Como G' contém no máximo $k_d + 1$ cliques, temos que $|V_C| \geq n/(k_d+1)$. Sendo $k_d < k$, conseqüentemente, $|V_C| \geq n/k$. Usando a

relação $n > 2(k+1) \cdot k$, obtemos

$$|V_C| > 2(k+1). \tag{5.1}$$

Como os vértices de V_C formam uma clique em G' e no máximo k_a novas arestas são adicionadas na transformação de G em G', existem em G no máximo k pares de vértices $\{v_i, v_j\}$, com $i < j$ e $v_i, v_j \in V_C$, que não estão conectados por uma aresta. Lembrando que, por hipótese, $|V_C| > 2(k+1)$ e $|\{\{v_i, v_j\} | v_i, v_j \in V_C, i < j, (v_i, v_j) \notin E\}| \leq k$, consideremos separadamente os casos em que $k_a = k$ e que $k_a < k$:

Caso 2.1: $k_a = k$. Então, $V_C = V$ (pois G é conexo e nenhuma aresta foi removida) e G' consiste de uma única clique. Como $k_a = k \geq 1$, existem dois vértices $v_i, v_j \in V$, com $i < j$, tais que $(v_i, v_j) \notin E$. Como V contém mais de $2(k+1)$ vértices, existem mais de $\binom{2(k+1)}{2}$ pares de vértices, sendo que no máximo k destes pares não estão conectados por uma aresta. Como $\{v_i, v_j\}$ é um destes pares, temos no máximo $k-1$ vértices não adjacentes a pelo menos um vértice dentre o par $\{v_i, v_j\}$. Dos mais de $2k$ vértices restantes (excluindo i e j), nos restam, então, pelo menos $k+1$ vizinhos comuns a v_i e v_j em G, levando ao fato de que a Regra 1 poderia ser aplicada; uma contradição à hipótese de que G é reduzido em relação à Regra 1.

Caso 2.2: $k_a < k$. Logo, podemos concluir que existem dois vértices $u \in V_C$ e $v \notin V_C$ tais que $(u, v) \in E$. Devido à inequação 5.1, existem mais de $2(k+1)$ vértices em V_C. Dentre eles, existem pelo menos $2(k+1) - k_a - 1$ vértices $u' \in V_C$ tais que $(u', u) \in E$ (pois, no pior caso, todas as k_a arestas adicionadas são incidentes a u). Além disso, existem no máximo k_d vértices $u' \in V_C$ tais que $(u', v) \in E$ (no pior caso, todas as k_d arestas unem v a algum vértice $u' \in V_C$). Conseqüentemente, temos pelo menos

$$2(k+1) - (k_a + k_d) - 1 = 2(k+1) - k - 1 = k + 1$$

vértices $u' \in V_C$ tais que $(u', u) \in E$ mas $(u', v) \notin E$. Isto implica que u tem pelo menos $k+1$ vizinhos em G que não são vizinhos a v, e a Regra 1 se aplicaria.

Em ambos os casos ($k_a = 0$ e $1 \leq k_a \leq k$), foi obtida uma contradição à hipótese de que G é reduzido em relação à Regra 1.

Em relação ao conjunto de arestas da componente conexa, obtemos uma contradição à hipótese de que $m > 2\binom{k+1}{2}k$ de uma forma análoga à feita para o conjunto de vértices.

Resumindo, o grafo reduzido contém no máximo $2(k^2 + k)$ vértices e no máximo $2\binom{k+1}{2}k = k^3 + k^2$ arestas, não existindo solução em caso contrário. O tempo de execução segue diretamente dos Lemas 5.2 e 5.3. □

5.4 Algoritmo de busca em árvore para o PA-M(k)

A idéia básica do algoritmo é identificar uma *tripla de conflito*, que consiste de três vértices, e ramificar a árvore em subcasos, a fim de desfazer o conflito, adicionando ou removendo arestas entre os três vértices considerados. Assim, são invocadas recursivamente chamadas ao algoritmo sobre instâncias simplificadas, uma vez que o valor do parâmetro é decrementado de pelo menos um. Antes de iniciar o algoritmo e depois de cada passo de ramificação, o núcleo do problema é computado conforme descrito na seção anterior. O tempo de execução do algoritmo é, então, determinado principalmente pelo tamanho da árvore de busca resultante. A seguir, será explanada uma estratégia de ramificação direta que conduz a uma árvore de busca com tamanho de $O(3^k)$. Posteriormente, será mostrada uma estratégia um pouco mais elaborada, responsável por uma árvore de busca cujo tamanho, no pior caso, é de $O(2.27^k)$.

5.4.1 Estratégia de ramificação básica

O ponto central da estratégia descrita nesta seção é o lema a seguir.

Lema 5.5 [11] *Um grafo $G = (V, E)$ consiste de cliques disjuntas se e somente se não existirem três vértices $u, v, w \in V$ tais que $(u, v) \in E$, $(u, w) \in E$, mas $(v, w) \notin E$.* □

Assim, se um dado grafo não consistir de cliques disjuntas, então podemos encontrar nele uma tripla de conflito e inserirmos ou removermos uma aresta, a fim de transformá-lo em cliques disjuntas. Como entrada do algoritmo, são fornecidos um grafo $G = (V, E)$ e um inteiro não negativo k. O procedimento informa como saída se G pode ser transformado em uma união de cliques disjuntas pela remoção ou adição de no máximo k arestas.

Algoritmo:

- Se o grafo G já for uma união de cliques disjuntas, então informe a solução e retorne.

- Caso contrário, se $k \leq 0$, então não é possível encontrar uma solução neste ramo da árvore de busca: Retorne.

- Caso contrário, identifique $u, v, w \in V$ tais que $(u, v) \in E$, $(u, w) \in E$, mas $(v, w) \notin E$. Recursivamente, chame o procedimento nas três instâncias a seguir, que consistem de grafos $G' = (V, E')$ com inteiro não negativo k' como especificado:

  (B1) $E' := E - (u, v)$ e $k' := k - 1$. Configurar $T[u, v] := \textit{proibida}$.

  (B2) $E' := E - (u, w)$ e $k' := k - 1$. Configurar $T[u, w] := \textit{proibida}$, $T[u, v] := \textit{permanente}$ e $T[v, w] := \textit{proibida}$.

  (B3) $E' := E + (v, w)$ e $k' := k - 1$. Configurar $T[u, v] := \textit{permanente}$, $T[u, w] := \textit{permanente}$ e $T[v, w] := \textit{permanente}$.

A partir de (B1), é possível obter 3 configurações possíveis, como ilustrado na Figura 5.6.

Considerando o simétrico de (B1), ou seja, $E' := E - (u, w)$ e $T[u, w] := \textit{proibida}$, obtemos também 3 configurações: os casos (a) e (b) da Figura 5.6 e o caso (B2). Por esta razão, podemos ignorar o simétrico de (B1) e representarmos diretamente o caso (B2).

A corretude deste procedimento é óbvia. Seu tempo de execução é garantido pela seguinte proposição:

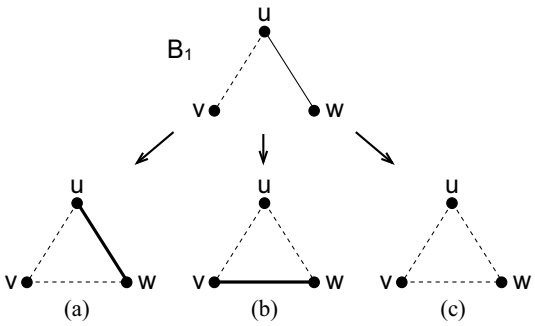

Figura 5.6: Configurações obtidas a partir de (B1). As linhas pontilhadas indicam arestas proibidas e as linhas em negrito, arestas permanentes.

Proposição 5.6 [11] *O PA-M(k) pode ser resolvido em um tempo de* $O(3^k \cdot k^2 + n^3)$.

Prova: O pré-processamento realizado no início, a fim de obter o núcleo do problema, pode ser feito em um tempo de $O(n^3)$. Posteriormente, é empregada uma árvore de busca, cujo tamanho é claramente limitado por $O(3^k)$. O fator k^2 é decorrente do custo computacional relacionado a cada nó da árvore: Primeiramente notemos que, em um passo de pré-processamento adicional, é possível configurar uma lista encadeada de todas as triplas de conflito, em um tempo de $O(n^3)$. Além disso, dentro de cada nó da árvore (exceto na raiz), uma aresta é removida ou adicionada, gerando, portanto, a necessidade de atualização das listas de conflito. De acordo com o Teorema 5.4, o grafo possui $O(k^2)$ vértices. Com um pequeno esforço, é possível verificar que a adição ou a deleção de uma aresta podem fazer com que surjam ou desapareçam no máximo $O(k^2)$ novas triplas de conflito. Usando técnicas de espalhamento combinadas com listas duplamente encadeadas para todas as triplas de conflito, podem-se realizar $O(k^2)$ atualizações nesta lista e na tabela em um tempo de $O(k^2)$. □

Corolário 5.7 [11] *O PA-M(k) pode ser resolvido em um tempo de* $O(3^k + n^3)$.

Prova: Segundo [11], no caso de um núcleo do problema de tamanho polinomial, obtido por um processo repetitivo de se gerar um núcleo durante

a execução do algoritmo de busca em árvore, sempre que possível, o fator polinomial no parâmetro k pode ser substituído por um fator constante. □

5.4.2 Melhorando a estratégia de ramificação

Nesta nova estratégia, uma tripla de conflito continua sendo identificada. Entretanto, passos de ramificação adicionais são verificados para todas as situações possíveis, baseados em um estudo caso a caso. A análise amortizada dos sucessivos passos fornecem, então, um melhor limite de pior caso para o tempo de execução. Inicialmente, distinguem-se três situações principais que podem ser analisadas, considerando-se uma tripla de conflito:

(C1) Os vértices v e w não possuem outros vizinhos comuns, ou seja, $\nexists x \in V, x \neq u$, tal que $(v, x) \in E$ e $(w, x) \in E$.

(C2) Os vértices v e w possuem um vizinho comum $x \neq u$, e $(u, x) \in E$.

(C3) Os vértices v e w possuem um vizinho comum $x \neq u$, e $(u, x) \notin E$.

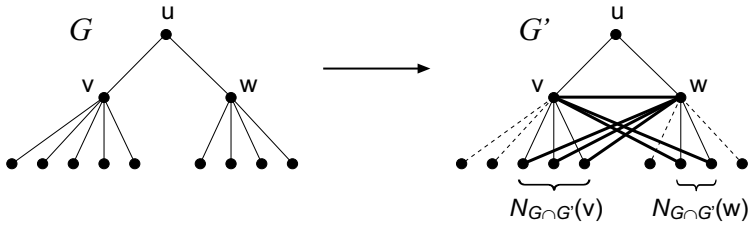

Figura 5.7: No caso (C1), a adição da aresta (v, w) não precisa ser considerada. Aqui, G é um grafo dado e G' é uma solução para o PA-M(k) aplicado a G, obtida pela adição da aresta (v, w). As linhas pontilhadas representam as arestas removidas na transformação de G em G' e as linhas em negrito representam as arestas adicionadas. A figura ilustra apenas parte dos grafos (em particular, das arestas) que são relevantes na argumentação.

Considerando o caso (C1), será mostrado pelo lema a seguir que a ramificação em apenas dois casos descritos na seção 5.4.1 – (B1) e (B2), é suficiente.

Lema 5.8 [11] *Dados um grafo $G = (V, E)$, um inteiro não negativo k e vértices $u, v, w \in V$, tais que $(u, v) \in E$, $(u, w) \in E$, mas $(v, w) \notin E$. Se v e w não compartilham nenhum vizinho comum (além de u), então, o caso de divisão (B3) não pode gerar uma solução melhor que as dos casos (B1) e (B2), podendo, portanto, ser omitido.*

Prova: Consideremos a solução G' do PA-M(k) para G onde a aresta (v, w) foi adicionada (ver Figura 5.7). A notação $N_{G \cap G'}(v)$ é usada para denotar o conjunto de vértices que são vizinhos a v em G e em G'. Sem perda de generalidade, assume-se que $|N_{G \cap G'}(w)| \leq |N_{G \cap G'}(v)|$. Podemos, então, construir um novo grafo G'' a partir de G', removendo todas as arestas adjacentes a w. Obviamente, G'' também é uma solução do PA-M(k) para G. Comparando o custo da transformação de $G \rightarrow G''$ com o de $G \rightarrow G'$, temos:

- -1 por não adicionar (v, w),
- $+1$ por remover (u, w),
- $-|N_{G \cap G'}(v)|$ por não adicionar todas as arestas (w, x), $x \in N_{G \cap G'}(v)$,
- $+|N_{G \cap G'}(w)|$ por remover todas as arestas (w, x), $x \in N_{G \cap G'}(w)$.

Os possíveis vértices que são vizinhos de w em G' mas não são vizinhos de w em G foram omitidos, uma vez que somente aumentariam o custo da transformação de $G \rightarrow G'$.

Logo, o custo da transformação de $G \rightarrow G''$ não é maior que o de $G \rightarrow G'$, onde o custo representa o número necessário de adições e deleções de arestas. □

Assim, quando o caso (C1) é identificado, a chamada recursiva é feita para os grafos $G_1 = (V, E - (u, v))$ e $G_2 = (V, E - (u, w))$, cada uma delas com o valor do parâmetro decrementado de um.

Para o caso (C2), como primeiro ramo temos o acréscimo da aresta (v, w). Como segundo e terceiro ramos, temos a remoção de (u, v) e de (u, w), respectivamente, como ilustrado na Figura 5.8.

No primeiro ramo, a aresta (v, w) é adicionada, rotulando o par de vértices como permanente, conforme ilustrado em ②. O custo deste subcaso é 1.

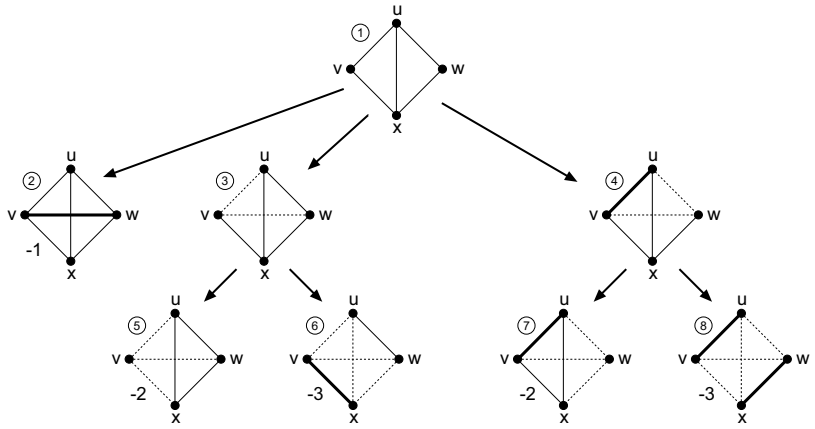

Figura 5.8: Ramificação do caso (C2). As linhas em negrito representam arestas permanentes, as pontilhadas representam arestas proibidas.

No segundo ramo, $\{v, w\}$ é configurado como proibida e a aresta (u, v) é removida, como ilustrado em ③ . Esta operação cria uma nova tripla de conflito: u, v e x. Para resolver este conflito, é realizada uma segunda ramificação. Como a adição de (u, v) é proibida, existem somente dois ramos a serem considerados:

– Remover (v, x), como ilustrado em ⑤. O custo é 2.

– Configurar $\{v, x\}$ como permanente (segundo (B2)) e remover (u, x). Devido à Regra 2 da seção 5.3, a aresta (w, x) também é removida, conforme ⑥. O custo deste subcaso é 3.

No terceiro ramo, configura-se $\{v, w\}$ como proibida e $\{u, v\}$ como permanente, removendo a aresta (u, w) (④). A resolução deste caso é simétrica à do caso anterior, originando dois ramos de custos 2 e 3, respectivamente.

Portanto, o vetor de ramificação obtido para o caso (C2) é $(1, 2, 3, 2, 3)$.

Para o caso (C3), a ramificação é feita conforme ilustrado na Figura 5.9.

No primeiro ramo, a aresta (u, v) é removida, como ilustrado em ② . O custo deste ramo é 1.

No segundo ramo, $\{u, v\}$ é configurado como permanente e a aresta (u, w) é removida, como em ③ . Devido à Regra 2, o par de vértices $\{v, w\}$ é

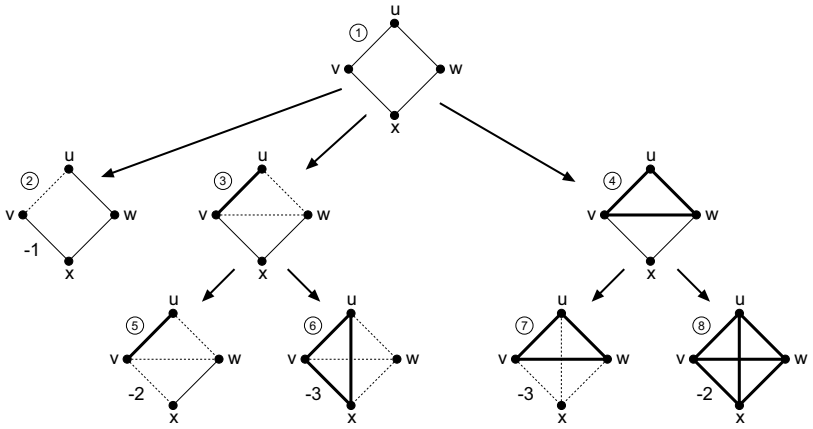

Figura 5.9: Ramificação do caso (C3).

configurado como proibida. Conseqüentemente, uma nova tripla de conflito u, v, x tem sua situação alterada, com a mudança de estado de um de seus pares de vértices. Como a remoção de (u, v) não é permitida, existem dois ramos possíveis para resolver este conflito:

– Remover (v, x), como em ⑤, com custo 2.

– Configurar $\{v, x\}$ como permanente. Isto implica que a aresta (u, x) tem que ser adicionada e (w, x) tem que ser removida, devido à Regra 2, como em ⑥. O custo deste ramo é 3.

No terceiro ramo, $\{u, v\}$ e $\{u, w\}$ são configurados como permanentes, e (v, w) é adicionada, como em ④. Os vértices u, w, x formam, então, mais uma tripla de conflito. Para resolvê-la sem remover (u, w), temos dois ramos:

– Remover (w, x), como em ⑦. Assim, a remoção da aresta (v, x) também é necessária, com custo total 3. Adicionalmente, o par $\{u, x\}$ é configurado como proibida.

– Adicionar (u, x), como ilustrado em ⑧, com custo 2. Adicionalmente, $\{u, x\}$ e $\{v, x\}$ tornam-se permanentes.

Portanto, o vetor de ramificação para o caso (C3) é $(1, 2, 3, 3, 2)$.

Em resumo, obtemos um refinamento para a ramificação, com um vetor de pior caso $(1, 2, 2, 3, 3)$, que resulta no número de ramificação 2.27. Como

o algoritmo recursivo pára caso o valor do parâmetro atinja um valor menor que zero, a árvore de busca obtida possui um tamanho de $O(2.27^k)$, como exposto no teorema a seguir.

Teorema 5.9 [11] *O PA-M(k) pode ser resolvido em um tempo de* $O(2.27^k + n^3)$. □

5.4.3 Melhorias heurísticas

As regras a seguir não alteram a complexidade de tempo de pior caso, uma vez que não há garantias de que alguma delas venha a ser aplicada. No entanto, elas podem ser usadas em uma implementação prática, já que são de custo computacional baixo e podem ajudar a reduzir significativamente o tamanho da árvore de busca.

Regra de ramificação

- Se $(u, v) \in E$ e u e v não possuem nenhum vizinho comum, são criados dois ramos: ou a aresta (u, v) é removida, ou todas as arestas adjacentes a u ou v (exceto (u, v)) são removidas, deixando o par $\{u, v\}$ como um 2-clique. Com uma argumentação similar a usada no Lema 5.8, é fácil ver que uma solução ótima pode ser encontrada em um destes dois ramos. Esta divisão normalmente é melhor que a do Teorema 5.9 (caso (C1)). Por exemplo, se u e v tiverem ambos grau 3, a divisão corresponde ao vetor de ramificação $(1, 4)$ e ao número de ramificação 1.39.

Regras de redução

Em alguns casos, nenhuma ramificação é necessária, e uma instância G com parâmetro k pode ser substituída diretamente por uma instância simplificada G' com parâmetro k'. A corretude das regras a seguir pode ser facilmente vista, através das regras anteriores e de argumentações simétricas.

Sejam u, v, w, x, y vértices distintos.

- Se $d(u) = d(v) = 1$ e $N(u) = N(v) = \{w\}$, como ilustrado na Figura 5.10 (a), então a aresta (u, w) é removida e configura-se $k' := k - 1$.

- Se $d(u) = 1$, $d(v) = 2$, $N(u) = \{v\}$ e $N(v) = \{u, w\}$ (caso (b) da Figura 5.10), então (v, w) é removida e $k' := k - 1$.
- Se $d(u) = 2$, $d(v) = d(w) = 3$ e $N(u) = \{v, w\}$, $N(v) = \{u, w, x\}$, $N(w) = \{u, v, y\}$ (caso (c) da Figura 5.10), então (v, x) e (w, y) são removidas e $k' := k - 2$.

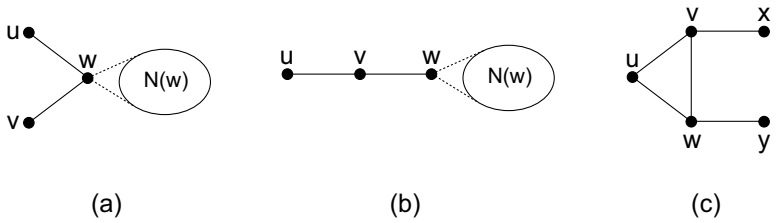

Figura 5.10: Regras heurísticas de redução.

Regra de descarte

Algumas ramificações da árvore de busca não precisam ser seguidas, ou porque elas não podem conduzir a uma solução, ou porque se sabe que, para qualquer solução a qual elas possam conduzir, é possível encontrar outra solução, pelo menos tão boa quanto ela.

- Sejam G_0 o grafo original de entrada e G o grafo no estado corrente do algoritmo. Se G contém um vértice v tal que $d_G(v) \geq 2d_{G_0}(v)$, então a ramificação corrente pode ser omitida, já que é certo encontrar uma solução ótima em outro ramo da árvore. A corretude desta regra vem do fato de que, para qualquer solução G' do PA-M(k) de G, é possível construir outra solução G'', removendo todas as arestas adjacentes a v em G'. Claramente, o custo de se transformar G_0 em G'' não é maior que o custo de se transformar G_0 em G'.

5.5 Algoritmo para o PA-R(k)

A partir do algoritmo de divisão em 3 ramos para o PA-M(k), exposto na seção 5.4.1, podemos obter um algoritmo com tempo de execução de

$O(2^k + n^3)$ para o PA-R(k). Dada uma tripla de conflito $u, v, w \in V$, a inserção de uma aresta não é permitida. Logo, são criados somente dois ramos: remove-se a aresta (u, v) ou a aresta (u, w). No decorrer desta seção, será mostrado que o número de ramificação pode ser melhorado de 2 para 1.77.

Como na seção 5.4.2, iniciamos o algoritmo pela identificação da tripla de conflito $u, v, w \in V$, com $(u, v) \in E$, $(u, w) \in E$, mas $(v, w) \notin E$, e distinguimos três casos:

(C1) Os vértices v e w não compartilham nenhum vizinho comum além de u, isto é, $\nexists x \in V, x \neq u$: $(v, x) \in E$ e $(w, x) \in E$.

(C2) Os vértices v e w possuem um vizinho comum $x \neq u$ e $(u, x) \notin E$.

(C3) Os vértices v e w possuem um vizinho comum $x \neq u$ e $(u, x) \in E$.

Considerando o caso (C1), distinguimos três subcasos:

(C1.1) Os vértices v e w não possuem outros vizinhos além de u:
Então, não é necessário realizar nenhuma ramificação; basta remover arbitrariamente uma das arestas, (u, v) ou (u, w), e resolver o conflito.

(C1.2) Um dos dois vértices, v ou w, possui um vizinho $x \neq u$, tal que $(u, x) \notin E$. Assumindo que v possua tal vizinho, ou seja, $(v, x) \in E$, $(w, x) \notin E$ e $(u, x) \notin E$, temos dois ramos:

- Remover a aresta (u, v). O custo deste ramo é 1.
- Configurar o par $\{u, v\}$ como permanente e remover a aresta (u, w). Conseqüentemente, a aresta (v, x) também tem que ser removida. O custo deste ramo é 2.

(C1.3) Todos os vizinhos de v e w também são vizinhos de u. Seja x um desses vizinhos. Assume-se que $(v, x) \in E$, $(w, x) \notin E$ e $(u, x) \in E$. São criados, então, dois ramos:

- Remover (u, w), com custo 1.

- Configurar $\{u, w\}$ como permanente e remover (u, v). Como não existe aresta entre w e x e $\{u, w\}$ é permanente, a aresta (u, x) também tem que ser removida. O custo deste ramo é 2.

Logo, o vetor de ramificação de pior caso para (C1) é $(1, 2)$.

Para o caso (C2), temos a divisão em dois ramos. No primeiro, removemos a aresta (u, v), com custo 1. No segundo ramo, configuramos $\{u, v\}$ como permanente e removemos (u, w). No entanto, esta operação alcança uma nova tripla de conflito: u, v, x. Como $\{u, v\}$ é permanente, a única solução para o conflito é remover (v, x). O custo deste ramo é 2.

Conseqüentemente, o vetor para o caso (C2) é $(1, 2)$.

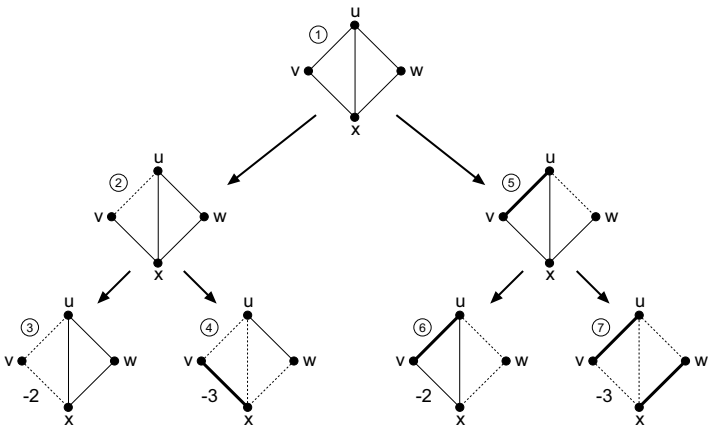

Figura 5.11: Ramificação do caso (C3) do algoritmo de busca em árvore para o PA-R(k). As linhas em negrito representam arestas permanentes, as pontilhadas representam arestas proibidas.

Para o caso (C3), é aplicada uma divisão em dois ramos, conforme ilustrado na Figura 5.11.

No primeiro ramo, a aresta (u, v) é removida (② na Figura 5.11). Isto cria uma nova tripla de conflito x, u, v. Para resolvê-lo, há duas formas:

– Remover (v, x), como em ③. O custo deste ramo é 2.

– Configurar $\{v, x\}$ como permanente e remover (u, x), conforme ④. Entretanto, isto implica que (w, x) também tem que ser removida. O custo é 3.

No segundo ramo, $\{u, v\}$ é configurado como permanente e a aresta (u, w) é removida, como em ⑤. Isto cria uma nova tripla de conflito x, u, w. Para resolver este conflito, há duas formas:

– Remover (w, x), como em ⑥. O custo deste ramo é 2.

– Configurar $\{w, x\}$ como permanente e remover a aresta (u, x) (⑦). Isto implica que (v, x) também tem que ser removida, a um custo total 3.

Portanto, o vetor de ramificação para (C3) é $(2, 3, 2, 3)$.

Comparando os três casos, temos que (C3) constitui o pior caso. Seu vetor corresponde ao número de ramificação 1.77. Analogamente ao Teorema 5.9, obtemos o seguinte teorema:

Teorema 5.10 [11] *O PA-R(k) pode ser resolvido em um tempo de* $O(1.77^k + n^3)$. □

Capítulo 6

Considerações finais

Há cerca de quinze anos, Downey e Fellows desenvolveram uma nova forma de analisar a complexidade de problemas computacionais, através de um refinamento na entrada do problema. Desde então, pesquisadores de todo o mundo vêm trabalhando intensivamente na busca de novos e importantes resultados para problemas que, segundo a teoria clássica de complexidade, são tidos como NP-completos. Este fato pode ser observado pela intensidade de publicações relacionadas à complexidade parametrizada, destacando-se soluções para problemas parametrizados e provas de intratabilidade. Por esta razão, ficaria inviável a realização de um estudo completo de todas as principais famílias da complexidade parametrizada, juntamente com exemplos de problemas pertencentes a tais famílias.

Esta dissertação teve como foco principal a classe FPT, onde procuramos exibi-la através de resultados obtidos para alguns problemas NP-completos. Seu desenvolvimento teve, portanto, o objetivo de fornecer conhecimentos básicos e necessários ao entendimento de dezenas de outros resultados de tratabilidade alcançados ao longo destes anos. Classificações de centenas de problemas parametrizados, bem como problemas para os quais ainda não se obteve classificação, podem ser encontrados, por exemplo, em [3] e [5].

No Capítulo 4, apresentamos um resultado de tratabilidade para o problema de cordalização(k). Outras classes foram pesquisadas para esta dissertação, incluindo uma superclasse dos grafos cordais: os *grafos fracamente cordais*. Um grafo G é fracamente cordal se e somente se todo ciclo de

comprimento > 4 em G e no seu complemento $\overline{G}$ possui uma corda. Entretanto, o problema de cordalização mínima para esta classe de grafos é polinomial, computável em um tempo de $O(n^2\overline{m}^2)$. Conseqüentemente, não há motivação para o desenvolvimento de sua versão parametrizada.

H. Kaplan, R. Shamir e R. E. Tarjan apresentaram também em [12] algoritmos tratáveis para o problema de completude parametrizada de duas subfamílias dos grafos cordais: *grafos fortemente cordais* e *grafos de intervalo formal*, sendo a segunda uma subfamília da primeira.

Para caracterizarmos os grafos fortemente cordais, precisamos da seguinte definição: Uma corda (v, w) de um ciclo par C é *ímpar* se os caminhos que conectam v e w em C contêm um número ímpar de arestas.

Um grafo G é fortemente cordal se e somente se G é cordal e todo ciclo par com comprimento pelo menos seis em G possui uma corda ímpar. Para esta subfamília dos cordais, foi desenvolvido um algoritmo tratável por parâmetro fixo que encontra todos os supergrafos fortemente cordais minimais de um dado grafo G pela adição de no máximo k arestas em um tempo de $O(8^{2k}m \log n)$.

Um grafo G é de intervalo formal se G é cordal e não contém nenhuma das três obstruções apresentadas na Figura 6.1 como subgrafo induzido.

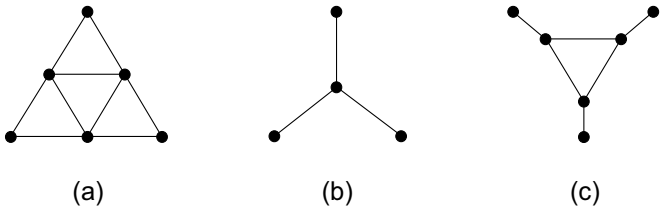

Figura 6.1: Obstruções para grafos cordais que não são de intervalo formal. (a) Tenda. (b) Garra. (c) Rede.

Para esta subfamília dos grafos fortemente cordais, foi apresentado um algoritmo tratável por parâmetro fixo que encontra todos os supergrafos de intervalo formal de um dado grafo G pela adição de no máximo k arestas em um tempo de $O(2^{4k}m)$.

Em 2002 [11], R. Shamir, R. Sharan e D. Tsur provaram a NP-completude dos problemas de p-aglomeração por modificação de arestas, para $p \geq 2$, e p-aglomeração por remoção de arestas, para $p \geq 3$. Nestes problemas, p denota o número de cliques que devem ser geradas pelo menor número de modificações (remoções) de arestas possível. Conseqüentemente, não há esperanças na tratabilidade por parâmetro fixo em relação a p, pois isto implicaria em $P = NP$. Considerando o problema de p-aglomeração por adição de arestas, Shamir et al. obtiveram um algoritmo com tempo de execução de $O(n^p)$.

Referências Bibliográficas

[1] An, X. *How many fill-ins can chordalize a graph?* Disponível em: `http://www.math.uwaterloo.ca/~xan/paper/chordal.doc`. Acesso em: 19 mar. 2003.

[2] Cai, L. *Fixed-parameter tractability of graph modification problems for hereditary properties*. Information Processing Letters, v. 58, p. 171-176, 1996.

[3] Cesati, M. *Compendium of Parameterized Problems*, 2001. Disponível em: `http://citeseer.ist.psu.edu/cesati01compendium.html`

[4] Cormen, T. H.; Leiserson, C. E.; Rivest, R. L. *Introduction to Algorithms*, 1ª edição. MIT Press, Cambridge, 1990.

[5] Downey, R. G.; Fellows, M. R. *Fixed-Parameter Tractability and Completeness I: Basic Results*. SIAM Journal on Computing, v. 24, n. 4, p. 873-921, 1995 .

[6] Downey, R. G.; Fellows, M. R. *Parameterized Complexity After (Almost) 10 Years: Review and Open Questions*, 1998. Disponível em: `http://citeseer.ist.psu.edu/565409.html`.

[7] Downey, R. G.; Fellows, M. R. *Parameterized Computational Feasibility*. P. Clote, J. Remmel (eds.): Feasible Mathematics {II}, p. 219-244. Birkhauser, 1995. Disponível em: `http://citeseer.ist.psu.edu/downey95parameterized.html`.

[8] Downey, R. G.; Fellows, M. R.; Niedermeier, R.; Rossmanith, P. *Parameterized Complexity*. International Conference and Research Center for

Computer Science, 2001. Disponível em:
http://www.dagstuhl.de/01311/Report/.

[9] Downey, R. G.; Fellows, M. R.; Stege, U. *Parameterized Complexity: A Framework for Systematically Confronting Computational Intractability*, 1998. Disponível em: http://citeseer.ist.psu.edu/561262.html.

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

[11] Gramm, J.; Guo, J.; Hüffner, F.; Niedermeier, R. *Graph-Modeled Data Clustering: Fixed-Parameter Algorithms for Clique Generation*. Lecture Notes in Computer Science, n. 2653, p. 108-119. Springer, 2003. Disponível em:
http://www-fs.informatik.uni-tuebingen.de/~hueffner/.

[12] Kaplan, H.; Shamir, R.; Tarjan, R. E. *Tractability of parameterized completion problems on chordal, strongly chordal and proper interval graphs*. SIAM Journal on Computing, v. 28, n. 5, p. 1906-1922, 1999.

[13] Natanzon, A.; Shamir, R.; Sharan, R. *A polynomial approximation algorithm for the minimum fill-in problem*. SIAM Journal on Computing, v. 30, n. 4, p. 1067-1079, 2000.

[14] Rose, D. J.; Tarjan, R. E.; Lueker, G. S. *Algorithmic aspects of vertex elimination on graphs*. SIAM Journal on Computing, v. 5, n. 2, p. 266-283, 1976.

[15] Tarjan, R. E.; Yannakakis, M. *Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs*. SIAM Journal on Computing, v. 13, n. 3, p. 566-579, 1984.

[16] Yannakakis, M. *Computing the minimum fill-in is NP-complete*. SIAM Journal on Algebraic Discrete Methods, v. 2, n. 1, p. 77-79, 1981.