Universidade Federal do ABC
Centro de Matemática, Computação e Cognição

Relatório Científico Final

Uma Introdução à Complexidade Parametrizada

Número do Processo FAPESP: 2021/07076-9

Período de Vigência da Bolsa: 01/11/2021 a 31/10/2022

Período coberto pelo presente Relatório: 01/04/2022 a 31/10/2022

Beneficiário:
Gustavo da Silva Teixeira

Responsável:
Carla Negri Lintzmayer

Maio/2023

Resumo

Quando tratamos de problemas computacionais, buscamos sempre desenvolver algoritmos eficientes que os resolvam, para todas as instâncias possíveis. O conceito de *eficiência* de um algoritmo normalmente refere-se à sua complexidade de tempo ser polinomial no tamanho da entrada. Porém, existem muitos problemas relevantes para os quais não há esperança de que existam algoritmos eficientes que encontrem soluções ótimas para todas as instâncias. Estes problemas constituem as classes denominadas NP-*difícil* e NP-*completo*.

Tais problemas, por muitas vezes terem aplicações importantes no mundo real, ainda necessitam de alguma solução e, entre as alternativas para lidar com eles, poderíamos pensar em algoritmos que, mesmo não sendo eficientes, devolvessem soluções ótimas com menos consumo de tempo. A teoria da complexidade parametrizada propõe uma alternativa neste sentido, estudando algoritmos que encontram soluções ótimas para todas as instâncias destes problemas em tempo exponencial, mas cuja complexidade exponencial não dependa de todo o tamanho da entrada, mas sim de um valor denominado *parâmetro*, que representa o tamanho de um determinado aspecto da entrada, e normalmente é bem menor que essa última. Problemas que admitem tais algoritmos são chamados de *tratáveis por parâmetro fixo* (FPT), e formam a classe de problemas que também denominamos FPT.

Esta iniciação científica teve por objetivo introduzir o candidato à teoria da complexidade parametrizada, fornecendo uma visão geral da área, com o estudo dos principais problemas envolvidos, das principais técnicas de desenvolvimento de algoritmos para lidar com problemas FPT e de resultados importantes obtidos na área até então.

Sumário

1 Introdução

Muitos problemas com os quais nos deparamos na vida real podem ser modelados como problemas computacionais e, consequentemente, necessitam de algoritmos para serem resolvidos. Porém, é sabido que a complexidade dos diversos problemas computacionais é variada, fazendo com que nem sempre seja possível alcançar de forma eficiente – geralmente em relação a tempo – soluções ótimas para todas as instâncias possíveis de um problema.

Quando falamos em *eficiência* em relação a tempo, normalmente estamos nos referindo a algoritmos que têm complexidade de tempo polinomial em função do tamanho da entrada. Assim, na teoria da complexidade computacional, destacam-se algumas classes de problemas, agrupando-os por complexidade de tempo: a classe P, que contém problemas que admitem um algoritmo que resolve qualquer instância em tempo polinomial, e as classes NP-difícil e NP-completo, que contêm problemas para os quais provavelmente não há algoritmo de tempo polinomial que resolva otimamente todas suas instâncias.

Uma das principais questões em aberto em Ciência da Computação é se as classes P e NP representam o mesmo conjunto, isto é, se $\mathsf{P} = \mathsf{NP}$, o que significaria que existem algoritmos de tempo polinomial que resolvem os problemas NP-completos, mas apenas ainda não conseguimos encontrá-los, ou se a classe P é um subconjunto próprio da classe NP, i.e., $\mathsf{P} \subset \mathsf{NP}$, o que significaria que ainda não encontramos algoritmos de tempo polinomial para resolver alguns problemas pelo fato de tais problemas serem realmente insolúveis em tempo polinomial.

Fato é que, ao nos depararmos com um problema do mundo real que pode ser modelado computacionalmente como um problema NP-difícil, ainda desejamos resolvê-lo otimamente para instâncias de interesse. Porém, um problema ser NP-difícil significa que não conseguiremos encontrar (*i*) soluções ótimas (no caso de problemas de otimização), (*ii*) para todas as instâncias, (*iii*) em tempo polinomial, a menos que $\mathsf{P} = \mathsf{NP}$. Dessa forma, teremos de abrir mão de pelo menos uma destas três propriedades.

Se abrirmos mão de resolver o problema otimamente (o que é possível ao lidarmos com problemas de otimização), poderíamos recorrer, por exemplo, a algoritmos de aproximação, que garantem que as soluções devolvidas estão a um fator multiplicativo das soluções ótimas, ou a métodos heurísticos, que são baseados em escolhas intuitivas relacionadas a propriedades do problema e, apesar de não garantirem um fator de aproximação, devolvem boas soluções na prática.

Se abrirmos mão de resolver todas as instâncias, podemos utilizar algoritmos polinomiais que devolvem soluções ótimas para um determinado conjunto de instâncias, que obedeçam certas propriedades. Porém, além destes resultados não serem tão simples de se obter, o

conjunto de instâncias para as quais conseguimos resolver ótima e polinomialmente nem sempre é de interesse para o problema no mundo real.

Se abrirmos mão de resolver em tempo polinomial, poderíamos tentar resolver todas as instâncias otimamente através de um algoritmo de tempo exponencial no tamanho da entrada (de força bruta, por exemplo), mas é sabido que este tipo de algoritmo é inviável em relação ao tempo de execução, que seria descrito como uma função $O(c^n)$, onde $c > 1$ é uma constante e n representa o tamanho da entrada. Sabendo que a função exponencial atinge valores muito altos a partir de entradas de tamanho relativamente pequeno, é inviável resolver o problema otimamente com este tipo de algoritmo para as instâncias nas quais normalmente estamos interessados.

A teoria da complexidade parametrizada, criada na década de 1990 por Downey e Fellows [13], propõe uma alternativa ao último dos três casos anteriores, estudando algoritmos que resolvem otimamente todas as instâncias de um problema NP-difícil em tempo exponencial, mas cuja complexidade de tempo seja descrita por uma função da forma $f(k) \cdot n^{O(1)}$, onde n é o tamanho da entrada, k é um número inteiro denominado *parâmetro* que representa alguma propriedade relacionada à entrada, e $f(k)$ é uma função no parâmetro k.

Geralmente, o valor do parâmetro k é muito menor que o tamanho da entrada, o que significa que um algoritmo com complexidade de tempo $f(k)n^{O(1)}$, apesar de também ser exponencial, tem tempo de execução muito menor que um algoritmo de complexidade $O(c^n)$. Além disso, encontrar um algoritmo com complexidade de tempo $f(k)n^{O(1)}$ nos leva a crer que a "dificuldade do problema" (aquilo que impede que obtenhamos um algoritmo polinomial no tamanho da entrada) esteja relacionada a este parâmetro k.

Problemas para os quais existem algoritmos com complexidade de tempo $f(k)n^{O(1)}$, sendo k um parâmetro e $f(k)$ uma função em k, são chamados de *tratáveis por parâmetro fixo*, ou FPT, e pertencem a uma classe de problemas denominada, também, FPT.

Este trabalho teve como objetivo estudar os principais aspectos da teoria da complexidade parametrizada, bem como os principais resultados e técnicas utilizadas nos problemas já provados pertencerem à classe FPT. Dada a importância desta teoria e classe de problemas para a tratabilidade de problemas NP-difíceis e NP-completos, esta pesquisa introduziu o candidato às principais ferramentas da complexidade parametrizada, de modo que seja possível, futuramente, aplicá-las a problemas específicos.

O restante do relatório está dividido como segue: a Seção 2 traz as definições básicas sobre as teorias de complexidade clássica e parametrizada, dando base para o estudo das técnicas que são abordadas posteriormente; a Seção 3 apresenta a técnica de kernelização e mostra sua importância para a teoria da complexidade parametrizada como um todo; a

Seção 4 apresenta a técnica das árvores de busca limitadas; a Seção 5 nos introduz aos algoritmos aleatorizados e mostra como eles podem ser usados para obter algoritmos FPT; a Seção 6 trata sobre *treewidth*, trazendo algumas definições essenciais sobre o tema nas Seções 6.1 e 6.2; a Seção 7 e a Subseção 7.1 apresentam, respectivamente, um formalismo lógico para descrever problemas em grafos e um teorema que utiliza tal formalismo para classificar problemas em grafos como FPT; a Seção 8 aborda as formas de demonstrar que determinados problemas não são tratáveis por parâmetro fixo, onde as Subseções 8.1 e 8.2 tratam das reduções entre problemas no contexto parametrizado e a Subseção 8.3 nos mostra que os problemas intratáveis por parâmetro fixo pertencem a uma hierarquia de dificuldade; a Seção 9 traz nossas considerações finais sobre o trabalho realizado no período de pesquisa; o Apêndice A traz aplicações de *kernelização*, com três exemplos básicos da técnica nas Subseções A.1 a A.3; o Apêndice A.4 apresenta a decomposição em coroa, um resultado em teoria dos grafos utilizado na obtenção de *kernels* para problemas parametrizados e cuja aplicação é exemplificada nas Subseções A.4.1 e A.4.2; o Apêndice A.5 apresenta o Lema do Girassol, um resultado importante sobre conjuntos, também utilizado em *kernelização*, como exemplificado na Subseção A.5.1; o Apêndice B traz exemplos de aplicação da técnica de árvores de busca limitadas; o Apêndice C mostra exemplos das técnicas de aleatorização para obtenção de algoritmos FPT, mais especificamente a codificação por cores e a separação aleatória; por fim, o Apêndice D traz as resoluções de alguns problemas estudados através de exercícios do livro, como forma de praticar as diferentes técnicas abordadas na pesquisa.

2 Definições

Nesta seção, apresentamos algumas definições importantes para entender a teoria da complexidade parametrizada, trazendo desde conceitos básicos, como a noção de um problema computacional e o que são suas instâncias, até conceitos mais complexos, como as definições das classes de complexidade na abordagem parametrizada.

Um *problema (computacional)* é uma questão a ser respondida, normalmente contendo vários *parâmetros*, ou *variáveis*, como *entrada*. O *enunciado* de um problema especifica, em termos gerais, a natureza de cada um de seus parâmetros, bem como a relação desejada entre entrada e saída, por meio de uma declaração de quais propriedades a *saída*, ou *solução*, deve ter. Uma *instância* de um problema é uma especificação de valores para cada um dos parâmetros do problema, respeitando as restrições impostas no enunciado.

Um *algoritmo*, é uma sequência finita e bem definida de *instruções* para resolver um problema. Para tanto, o algoritmo toma como entrada uma instância do problema, executa uma sequência de instruções e produz um valor como saída, de modo a solucionar a questão

proposta no enunciado. Intuitivamente, podemos pensar em um algoritmo como um programa de computador, escrito em alguma linguagem de programação precisa.

Um *problema de decisão* Q é um problema para o qual as duas únicas soluções possíveis são SIM ou NÃO. Assim, podemos dizer que Q consiste em um conjunto D_Q de instâncias e um conjunto $Y_Q \subseteq D_Q$ de *instâncias* SIM – como denominaremos instâncias para as quais a solução é SIM.

Um *problema de otimização* Q consiste em um conjunto D_Q de instâncias, um conjunto S_Q de soluções e uma função $f\colon S_Q \to \mathbb{R}$, denominada *função de custo*. Para cada instância $I \in D_Q$, denotaremos por $S_Q[I]$ o conjunto de *soluções viáveis* para I, isto é, o conjunto das soluções obtidas a partir de I que satisfaçam todas as restrições do enunciado do problema de otimização. Uma *solução ótima* para uma instância I é uma solução $S \in S_Q[I]$ tal que $f(S)$ é máximo ou mínimo, a depender do objetivo do problema. Problemas em que buscamos maximizar $f(S)$ são chamados de *problemas de maximização*. Analogamente, problemas em que buscamos minimizar $f(S)$ são chamados *problemas de minimização*.

É importante atentarmos ao fato de que todos os problemas tratados em nossa pesquisa são de decisão, mas que, para alguns resultados e comentários posteriores, é interessante conhecer o conceito de problema de otimização. Normalmente, podemos expressar um determinado problema de otimização como um problema de decisão relacionado, impondo um limite para o valor a ser otimizado. Dessa forma, a questão a ser respondida no enunciado do problema deixa de ser "qual o valor máximo/mínimo $f(S)$ de uma determinada solução S para a instância I?" e passa a ser "o valor $f(S)$ de uma determinada solução S para a instância I é no máximo/mínimo um dado valor k?", tendo esta última questão apenas duas respostas possíveis, SIM ou NÃO.

Se Q é um problema de decisão, dizemos que um algoritmo $\mathcal{A}$ *resolve* Q se, para qualquer instância I de Q, $\mathcal{A}$ devolve SIM se e somente se I é uma instância SIM, isto é, se $I \in Y_Q$. Se Q é um problema de otimização, dizemos que um algoritmo $\mathcal{A}$ *resolve* Q se, para qualquer instância I de Q, $\mathcal{A}$ devolve uma solução ótima para I. Denotamos por $\mathcal{A}(I)$ a saída/solução de um algoritmo $\mathcal{A}$ para uma instância I de um problema.

Geralmente, estamos interessados em encontrar um algoritmo que, dada uma instância, consuma menos tempo para devolver uma solução. O *tempo de execução* de um algoritmo é a quantidade de *passos básicos* (ou instruções) executados por ele sobre uma instância de entrada. Como passos básicos, consideramos instruções tais como atribuições de valor a variáveis, operações aritméticas, operações relacionais etc. Intuitivamente, o *tamanho da entrada* de um problema é a quantidade de dados necessária para descrever a instância. Por exemplo, se a entrada do problema é uma sequência de n números reais, o tamanho da entrada pode ser medido por n; se a entrada for um grafo de n vértices e m arestas, podemos

medir o tamanho da entrada por $n + m$; e assim por diante. O tempo de execução de um algoritmo tende a crescer conforme o tamanho da entrada cresce e usamos uma função $f(n)$ para descrever o tempo de execução do algoritmo para instâncias de tamanho n.

Sejam as funções $f, g \colon \mathbb{N} \to \mathbb{R}^+$. Dizemos que $f(n) = O(g(n))$ se existem inteiros positivos c e n_0 tais que $f(n) \leq c \cdot g(n)$ para todo $n \geq n_0$.

Um *algoritmo de tempo polinomial* é definido como um algoritmo cuja função de complexidade de tempo é $O(p(n))$, para alguma função polinomial p, onde n denota o tamanho da entrada. Um algoritmo é dito *eficiente* se, no pior caso, tem complexidade de tempo polinomial no tamanho da entrada. Um problema é dito *tratável em tempo polinomial* se admite um algoritmo eficiente; caso contrário, o problema é dito *intratável em tempo polinomial*.

Um problema de decisão Q pertence à classe P se e somente se Q pode ser resolvido por um algoritmo eficiente.

Dados um problema de decisão Q e um algoritmo $\mathcal{A}$, dizemos que $\mathcal{A}$ é um *algoritmo verificador* se: (i) para toda instância I de Q cuja a resposta é SIM, existe um conjunto de dados D tal que $\mathcal{A}(I, D)$ devolve SIM; e (ii) para toda instância I de Q cuja a resposta é NÃO, para qualquer conjunto de dados D, vale que $\mathcal{A}(I, D)$ devolve NÃO. Chamamos o conjunto de dados D, que atenda a estas duas condições, de *certificado positivo*.

Dizemos que um problema de decisão Q pertence à classe NP se existe um algoritmo verificador para Q que aceita um certificado positivo em tempo polinomial.

Dados dois problemas de decisão Q e Q', dizemos que Q *se reduz a* Q *em tempo polinomial*, o que denotamos por $Q \propto Q'$, se existe um algoritmo que, dada uma instância I de Q, constrói uma instância I' de Q' em tempo polinomial no tamanho de I, tal que I' é uma instância SIM se e somente se I é uma instância SIM.

Um problema Q' é NP-difícil se para todo problema $Q \in \mathsf{NP}$, $Q \propto Q'$. Se Q' é NP-difícil e também pertence a NP, então Q' é NP-completo.

É fácil notar que $Q \in \mathsf{P}$ implica em $Q \in \mathsf{NP}$. Além disso, note que se um único problema NP-difícil puder ser resolvido em tempo polinomial, então todo problema em NP pode ser resolvido em tempo polinomial. Se existe um problema em NP que não pode ser resolvido em tempo polinomial, então nenhum problema NP-difícil pode ser resolvido em tempo polinomial. Portanto, um problema NP-completo Q possui a seguinte propriedade: $Q \in \mathsf{P}$ se e somente se $\mathsf{P} = \mathsf{NP}$. Descobrir se $\mathsf{P} = \mathsf{NP}$ é uma das mais importantes questões em aberto em Ciência da Computação.

2.1 Problemas como linguagens e algoritmos como máquinas de Turing

Nesta seção, trazemos definições mais formais sobre problemas e algoritmos, que são equivalentes às noções intuitivas mas que nos ajudarão a entender melhor alguns resultados e notações ao longo do relatório. Tais definições foram baseadas no clássico livro de Sipser [27], que consideramos trazer uma abordagem bem intuitiva para o assunto.

Um *alfabeto* é um conjunto finito e não-vazio cujos elementos são chamados de *símbolos*. Dado um alfabeto Σ, uma *cadeia* (*sobre* Σ), é uma sequência $x_1x_2\ldots x_n$, onde $x_i \in \Sigma$ para cada índice $1 \leq i \leq n$, isto é, uma cadeia é uma sequência finita de símbolos de Σ. O *comprimento* $|\omega|$ de uma cadeia ω é a quantidade de símbolos de ω. A *cadeia vazia*, denotada por ε, é a cadeia de comprimento zero.

Dado um alfabeto Σ, denotamos por Σ^k o conjunto de todas as cadeias de comprimento $k \geq 0$ sobre Σ. O *fecho de Kleene*, denotado por Σ^*, é o conjunto de todas as cadeias sobre Σ, isto é, $\Sigma^* = \bigcup_{k=0}^{\infty} \Sigma^k$. Assim, podemos denotar uma cadeia ω sobre o alfabeto Σ simplesmente por $\omega \in \Sigma^*$.

Uma *linguagem* L sobre um alfabeto Σ é um conjunto de cadeias sobre Σ, ou seja, $L \subseteq \Sigma^*$. A questão principal da qual trataremos é: "dadas uma linguagem $L \subseteq \Sigma^*$ e uma cadeia $\omega \in \Sigma$, podemos afirmar que $\omega \in L$?". Note que qualquer problema de decisão pode ser descrito como uma linguagem $L \subseteq \Sigma^*$ que contém as cadeias que são codificações de suas instâncias Sim, considerando algum *esquema de codificação*. Assim, denotaremos por $\langle I \rangle \subseteq \Sigma^*$ a codificação em cadeia de uma instância I para um problema. Dada uma codificação $\langle I \rangle$ de uma instância I passada como entrada, o *tamanho da entrada* será definido como $|\langle I \rangle|$, isto é, o número de símbolos necessários para codificar I como cadeia. Por exemplo, seja $\langle C, k \rangle \subseteq \Sigma^*$ a codificação em cadeia de um conjunto C de números naturais e um natural k. O problema de verificar se k é elemento de C (um problema de busca simples), pode ser descrito como a linguagem $L = \{\langle C, k \rangle : k \in C\} \subseteq \Sigma^*$.

Uma *Máquina de Turing (MT) determinística* é definida como uma 7-upla $(Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R)$ em que:

- Q é um conjunto finito de estados;
- Σ é o alfabeto de entrada, tal que o chamado *símbolo branco* $\sqcup$ não pertence a Σ;
- Γ é o alfabeto da fita, tal que $\Sigma \subseteq \Gamma$ e $\sqcup \in \Gamma$;
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{D, E\}$ é a função de transição, onde D e E representam a movimentação da cabeça da máquina para a direita ou para a esquerda, respectivamente;
- $q_0 \in Q$ é o estado inicial;

- $q_A \in Q$ é o estado de aceitação;
- $q_R \in Q$ é o estado de rejeição, com $q_R \neq q_A$.

Para definir uma *MT não-determinística*, alteramos a definição da função de transição δ para $\delta\colon (Q \cup \{\varepsilon\}) \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{D, E\})$, onde $\mathcal{P}(X)$ denota o conjunto das partes do conjunto X.

Uma configuração de uma MT determinística $M = (Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R)$ é descrita por $\alpha q \beta$, onde $\alpha, \beta \in \Gamma^*$, a concatenação $\alpha\beta$ é o conteúdo da fita, $q \in Q$, e a cabeça da máquina está sobre o primeiro símbolo de β. Seja $M = (Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R)$ uma MT e sejam $x, y, z \in \Gamma$, $\alpha, \beta \in \Gamma^*$ e $q_i, q_j \in Q$. Se $\delta(q_i, y) = (q_j, z, E)$, então $\alpha x q_i y \beta \vdash \alpha q_j x z \beta$ é um movimento de M, onde o símbolo $\vdash$ denota que a transição faz com que a máquina vá da configuração $\alpha x q_i y \beta$ para a configuração $\alpha q_j x z \beta$. De forma análoga, se $\delta(q_i, y) = (q_j, z, D)$, então $\alpha x q_i y \beta \vdash \alpha x z q_j \beta$ é um movimento de M. Como estamos definindo uma MT com a fita limitada à esquerda, há um caso especial de seu movimento: $q_i y \beta \vdash q_j z \beta$ quando $\delta(q_i, y) = (q_j, z, E)$. Isto significa que se a cabeça da MT estiver na posição mais à esquerda da fita, um movimento para a esquerda mantêm a cabeça na mesma posição. Podemos denotar por $\vdash^*$ uma sequência de zero ou mais movimentos em M.

Seja $M = (Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R)$ uma MT e $\omega \in \Sigma^*$. Dizemos que M *aceita* ω se $q_0\omega \vdash^* \alpha q_A \beta$. Analogamente, M *rejeita* ω se $q_0\omega \vdash^* \alpha q_R \beta$. Dizemos que M *decide* ω se $q_0\omega \vdash^* \alpha q_A \beta$ ou $q_0\omega \vdash^* \alpha q_R \beta$. Se uma MT nunca entrar em um estado de aceitação ou de rejeição, então dizemos que ela *entrou em loop* ou que ela *não para*.

No caso das máquinas de Turing não-determinísticas, podemos dizer que elas permitem que um único movimento leve a mais de um estado, de modo que a máquina pode estar com várias configurações em um mesmo momento. Com essa diferença, a sequência de configurações de uma MT determinística ao computar uma certa cadeia pode ser vista como um caminho, pois cada movimento permite ir da configuração atual a exatamente uma outra; por outro lado, a sequência de configurações de uma MT não-determinística pode ser vista como uma árvore, pois cada movimento permite ir de cada uma das configurações atuais a várias outras. Se algum ramo da árvore que representa a computação de uma máquina de Turing não-determinística leva ao estado de aceitação (resp. rejeição), a máquina aceita (resp. rejeita) a cadeia de entrada. Assim como no caso determinístico, se uma MT não-determinística nunca entrar em um estado de aceitação ou rejeição, ela entra em *loop*.

Se $L \subseteq \Sigma^*$ é o conjunto de cadeias que uma MT M aceita, chamamos L de *linguagem de* M ou *linguagem reconhecida por* M. Dizemos que uma MT M *reconhece* L se $L = \{\omega \in \Sigma^*\colon M \text{ aceita } \omega\}$. Dizemos que uma MT M *decide* L se, para toda $\omega \in \Sigma^*$, M decide ω.

Uma linguagem L é chamada de *Turing-decidível*, ou simplesmente *decidível* (ou ainda *recursiva*), se existe uma MT que a decide, isto é, dada $\omega \in \Sigma^*$, a MT decide se $\omega \in L$

ou $\omega \notin L$. Caso L não seja decidível, dizemos que L é *indecidível*. Uma linguagem L é chamada de *Turing-reconhecível*, ou simplesmente reconhecível (ou ainda *recursivamente enumerável*), se existe uma MT que a reconhece, isto é, dada $\omega \in \Sigma^*$, a MT a aceita ω caso $\omega \in L$, e rejeita ω ou entra em loop caso $\omega \notin L$. Caso L não seja reconhecível, dizemos que L é *irreconhecível*. Claramente, toda linguagem decidível é também reconhecível, mas não podemos afirmar contrário.

Uma função $f\colon \Sigma^* \to \Sigma^*$ é *computável* se alguma máquina de Turing, ao começar com uma cadeia ω na fita, *para* contendo $f(\omega)$ na fita. Note que existem *funções não-computáveis* e elas geralmente estão relacionadas a linguagens/problemas indecidíveis, como veremos a seguir. Um exemplo clássico de problema indecidível é o Problema da Parada, que tem como entrada uma MT M e uma cadeia ω o objetivo é decidir se M para quando executada sobre ω, ou seja, tal problema corresponde à linguagem $L = \{\langle M, \omega\rangle\colon M \text{ é uma MT que para sobre a cadeia } \omega\}$.

No contexto do Problema da Parada, seja $H(\langle M, \omega\rangle)$ uma função que recebe uma MT M e uma cadeia ω e devolve 1 se M entra em loop sobre ω, ou devolve 0 se M para sobre ω. Uma vez que o Problema da Parada é indecidível, a função H não pode ser computável, pois não existe uma MT que garanta que, para a entrada $\langle M, \omega\rangle$, a MT parará com $H(\langle M, \omega\rangle)$ na fita.

A importância de tais definições e das máquinas de Turing reside no fato de que o poder computacional dessas máquinas é equivalente ao poder dos algoritmos na noção intuitiva apresentada anteriormente. Assim, a existência de um algoritmo que resolve um problema é equivalente à existência de uma máquina de Turing que decide o problema descrito como linguagem. Além disso, sempre que nos referirmos a funções computáveis ao longo do relatório, queremos dizer que existe um algoritmo ou máquina de Turing decidível que realiza um procedimento correspondente à função computável em questão.

Uma linguagem A é redutível a uma linguagem B se existe uma função computável $f\colon \Sigma^* \to \Sigma^*$ tal que $\omega \in A$ se e somente se $f(\omega) \in B$, para toda $\omega \in \Sigma^*$.

Seja M uma máquina de Turing determinística decisora. O tempo de execução de M é descrito pela função $f\colon \mathbb{N} \to \mathbb{N}$ tal que $f(n)$ é o número máximo de movimentos que M executa sobre entradas de comprimento n. Seja N uma máquina de Turing não-determinística decisora. O tempo de execução de N é descrito pela função $f\colon \mathbb{N} \to \mathbb{N}$ tal que $f(n)$ é o número máximo de movimentos que N executa sobre qualquer ramo de sua árvore de computação sobre entradas de comprimento n.

A classe P é a classe de linguagens que são decidíveis em tempo polinomial por uma máquina de Turing determinística de fita única.

A classe NP é a classe de linguagens que são decidíveis em tempo polinomial por uma

máquina de Turing não-determinística.

2.2 Problemas parametrizados

A seguir, trazemos algumas definições formais sobre problemas parametrizados e algumas classes de problemas envolvidas na teoria da complexidade parametrizada.

Um *problema parametrizado* é uma linguagem $L \subseteq \Sigma^* \times \mathbb{N}$, onde Σ é um alfabeto finito e fixo. Para uma instância $(x, k) \in \Sigma^* \times \mathbb{N}$, o natural k é chamado de *parâmetro*. O tamanho de uma instância (x, k) de um problema parametrizado é definido como $|(x, k)| = |x| + k$. Uma interpretação dessa convenção é que, quando dado ao algoritmo na entrada, o parâmetro k é codificado em unário.

Um problema parametrizado $L \subseteq \Sigma^* \times \mathbb{N}$ é chamado de *tratável por parâmetro fixo* (FPT) se existe um algoritmo $\mathcal{A}$ (chamado de *algoritmo de parâmetro fixo*), uma função computável $f : \mathbb{N} \to \mathbb{N}$, e uma constante c tais que, dado $(x, k) \in \Sigma^* \times \mathbb{N}$, o algoritmo $\mathcal{A}$ decide corretamente se $(x, k) \in L$ em tempo limitado por $f(k) \cdot |(x, k)|^c$. A classe de complexidade contendo todos os problemas tratáveis por parâmetro fixo é chamada de *classe* FPT.

Um problema parametrizado $L \subseteq \Sigma^* \times \mathbb{N}$ pertence à classe XP se existir um algoritmo $\mathcal{A}$ e duas funções computáveis $f, g : \mathbb{N} \to \mathbb{N}$ tais que, dado $(x, k) \in \Sigma^* \times \mathbb{N}$, o algoritmo $\mathcal{A}$ decide corretamente se $(x, k) \in L$ em tempo limitado por $f(k) \cdot |(x, k)|^{g(k)}$.

Podemos generalizar as definições de problema parametrizado e das classes FPT e XP para que englobem vários parâmetros, de modo que k seja não apenas um inteiro não negativo, mas um vetor de d inteiros não negativos, para alguma constante fixa d. Neste caso, as funções f e g nas definições das classes de complexidade FPT e XP podem depender de todos esses parâmetros.

Utilizaremos o termo "função FPT" para denotar uma função da forma $f(k) \cdot n^{O(1)}$, ou seja, um produto de uma função computável f do parâmetro k e um polinômio $n^{O(1)}$ no tamanho de entrada $n = |(x, k)|$. O termo "tempo FPT" se refere a um tempo de execução descrito por uma função FPT, e o termo "algoritmo FPT" se refere a um algoritmo com tempo FPT. Tais termos são análogos a "função polinomial", "tempo polinomial" e "algoritmo polinomial" na teoria clássica. O mesmo vale para os termos "função XP", "tempo XP" e "algoritmo XP", onde uma função XP tem a forma $f(k) \cdot n^{g(k)}$, para f e n definidos como acima e sendo g uma função computável do parâmetro k.

2.3 Algumas definições em Teoria dos Grafos

Um *grafo (simples)* é definido por um par (V, E), onde V é um conjunto finito de elementos, chamados *vértices*, e E é um conjunto de pares não ordenados dos elementos de V, chamados

arestas. Também utilizaremos $V(G)$ e $E(G)$ para denotar, respectivamente, o conjunto de vértices e o conjunto de arestas de um grafo G. No que segue, sejam G e H grafos simples.

Denotaremos a aresta formada pelos vértices $u, v \in V(G)$ por uv (ou vu), representando o par não ordenado $\{u, v\} \in E(G)$. Os vértices u e v são *extremos* da aresta uv, a aresta uv *incide* em u e em v, e os vértices u e v são *vizinhos* (ou *adjacentes*).

Dado $v \in V(G)$, o *grau* de v é a quantidade de vizinhos de v e é denotado por $d_G(v)$ (ou apenas $d(v)$, quando for claro a qual grafo estamos nos referindo). O *grau máximo* do grafo G, denotado por $\Delta(G)$, é definido como $\max\{d(v) \colon v \in V(G)\}$. De forma análoga, o *grau mínimo* do grafo G, denotado por $\delta(G)$, é definido como $\min\{d(v) \colon v \in V(G)\}$.

O conjunto formado por todos os vizinhos de um vértice v é denominado *vizinhança* de v e é denotado por $N_G(v)$ (ou apenas $N(v)$, quando for claro a qual grafo estamos nos referindo).

Dizemos que H é *subgrafo* de G se $V(H) \subseteq V(G)$ e $E(H) \subseteq E(G)$, e denotamos essa relação por $H \subseteq G$.

Chamamos de *passeio* em G uma sequência de vértices $W = (v_0, v_1, \ldots, v_k)$ tal que $v_i v_{i+1} \in E(G)$, para $0 \leq i < k$, e podemos dizer, também, que W contém tais arestas $v_i v_{i+1}$, com $0 \leq i < k$. Chamamos v_0 de *origem* ou *vértice inicial* de W e v_k de *término* ou *vértice final* de W. Os vértices $v_1, \ldots, v_{k-1}$ são chamados de *vértices internos* de W. Um passeio W é dito *fechado* se $v_0 = v_k$.

Seja W um passeio. Se W não repete arestas, chamamos W de *trilha*. Se W é um passeio fechado e não repete arestas, chamamos W de *trilha fechada*. Se W não repete vértices, chamamos W de *caminho*. Se W é um passeio fechado com pelo menos três arestas e que não repete vértices internos, chamamos W de *ciclo*.

Se para qualquer par $u, v \in V(G)$ existe um passeio com início em u e término em v, então G é um grafo *conexo*.

Se $H \subseteq G$, dizemos que H é *maximal* em relação a uma propriedade P se não existe $H' \subseteq G$ tal que $H \subset H'$ e H' possui a propriedade P.

Se $H \subseteq G$ e H é maximal em relação à conexidade, então H é uma *componente (conexa)* de G. Um grafo que não é conexo é composto por várias componentes conexas. Se um grafo G é conexo, então $|E(G)| \geq |V(G)| - 1$. Se G não é conexo, então $|E(G)| \geq |V(G)| - c$, onde c é a quantidade de componentes conexas de G.

Se não existe nenhum ciclo em G, então G é uma *floresta*. Se G não possui ciclos e é conexo, então G é uma *árvore*. Um vértice v em uma floresta F é chamado de *folha* se $d_F(v) = 1$. Todo vértice que não é folha em uma floresta é chamado de *vértice interno*.

3 *Kernelização*

A primeira técnica estudada é denominada *kernelização* (ou *redução a um kernel*) e consiste em uma técnica de pré-processamento cujo objetivo é resolver de forma eficiente algumas partes da instância de um problema NP-difícil, a fim de reduzi-la a uma estrutura central, chamada de *kernel*, que pode ser vista como a parte computacionalmente difícil do problema.

Dada uma instância I para um problema parametrizado Q, um *kernel* para Q é uma instância I' para Q, *equivalente* à instância original, mas de tamanho menor. Por *instância equivalente* queremos dizer que I é SIM se e somente se I' é SIM. Uma vez que conseguirmos reduzir uma instância a um *kernel* em tempo polinomial, podemos utilizar um algoritmo exponencial (de força bruta, por exemplo) para resolvê-lo. Se a *kernelização* leva tempo polinomial e o tamanho $|I'|$ do *kernel* obtido é limitado superiormente por uma função computável $f(k)$, dependente apenas do parâmetro k do problema, o tempo total gasto entre a *kernelização* e a obtenção de uma solução para I' será FPT, como veremos em um teorema adiante.

Seja Q um problema parametrizado cuja instância $I = (x, k)$ consiste na entrada x codificada como uma cadeia sobre um alfabeto Σ e k é um número natural definido como parâmetro, com $|I| = |(x, k)| = |x| + k$. Uma *regra de redução* para Q é uma função $\phi : \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$ que tem como entrada uma instância I de Q e gera uma *instância equivalente* $I' = (x', k')$, tal que $\phi(x, k)$ é uma função computável em tempo polinomial, dependente de x e de k. Chamaremos de *segurança* da regra de redução a garantia de que a instância menor gerada por ela seja equivalente à instância original. Dessa forma, a ideia é projetar um algoritmo de pré-processamento, que aplica recursivamente várias regras de redução para que o tamanho da instância seja o mínimo possível.

Vejamos, então, como podemos transformar uma instância em uma instância equivalente de tamanho menor. Algoritmos de *kernelização* são algoritmos de pré-processamento cujo tamanho da saída é limitado por uma função computável do parâmetro do problema. Formalmente, um *algoritmo de kernelização* $\mathcal{K}$ para um problema parametrizado Q é um algoritmo que, dada uma instância $I = (x, k)$ de Q, executa em tempo polinomial em $|I| = |x| + k$ e, ou decide Q, devolvendo uma solução SIM/NÃO para I, ou devolve uma *instância equivalente reduzida* $I' = (x', k')$ de Q, o que significa que existe uma função computável g, dependente apenas de k, tal que $|I'| = |x'| + k' \leq g(k)$. Se a função g é uma função polinomial (resp. linear) do parâmetro, então dizemos que Q admite um *kernel polinomial (resp. linear)*. Nos casos em que um algoritmo de *kernelização* devolve uma instância equivalente reduzida I', chamamos I' de *kernel* de Q.

Como vemos na definição anterior, existem situações, como em alguns exemplos adiante,

em que o algoritmo de *kernelização* consegue resolver completamente a instância do problema, isto é, decidir se a instância é SIM ou NÃO. Por este motivo, permitimos que as regras de redução possam devolver tanto uma resposta SIM/NÃO para o problema quanto uma instância equivalente reduzida. Além disso, se existe um algoritmo de *kernelização* para um problema parametrizado Q, dizemos que Q *admite um algoritmo de kernelização* ou que Q *admite um kernel*.

A partir da definição de algoritmo de *kernelização*, podemos obter o seguinte lema.

Lema 1 ([7])**.** *Seja Q um problema parametrizado. Se Q admite um algoritmo de kernelização, então Q é FPT.*

Demonstração. Seja $I = (x, k)$ uma instância de Q, com $|I| = |(x, k)| = |x| + k$, e suponha que existam um algoritmo de *kernelização* $\mathcal{K}$ para Q e uma constante c tais que ou $\mathcal{K}$ decide I em tempo $|I|^c$ ou $\mathcal{K}$ devolve uma instância equivalente reduzida $I' = (x', k')$ no mesmo tempo.

Se $\mathcal{K}$ decide I, o tempo total para encontrar uma solução é $|I|^c = |(x, k)|^c$, que é uma função FPT. Se $\mathcal{K}$ devolve uma instância reduzida I', podemos executar um algoritmo de força bruta $\mathcal{B}$ em I', de tempo exponencial em $|I'|$. Nesse caso, como $|I'| \leq g(k)$, para uma função computável g dependente apenas do parâmetro k, o tempo de execução do algoritmo de força bruta $\mathcal{B}$ também será descrito por uma função (exponencial) dependente apenas do parâmetro k, digamos $h(k)$. Assim, o tempo total da execução $\mathcal{K}$ em I, devolvendo I', seguida da execução de $\mathcal{B}$ em I' será descrito por uma função da forma $|(x, k)|^c + h(k)$, que é uma função FPT. □

Assim, vimos que se um problema admite um algoritmo de kernelização, então ele pertence à classe FPT. O resultado surpreendente, que mostra a importância da técnica de *kernelização* para a teoria da complexidade parametrizada, é que a recíproca também é verdadeira, como vemos no seguinte teorema.

Teorema 2 ([7])**.** *Um problema parametrizado Q é FPT se e somente se Q admite um algoritmo de kernelização.*

Demonstração. ($\Leftarrow$) Provado no Lema 1.

($\Rightarrow$) Seja Q um problema parametrizado FPT com instâncias $I = (x, k)$, com $|I| = |(x, k)| = |x| + k$. Uma vez que Q é FPT, existem um algoritmo $\mathcal{A}$, uma constante c e uma função computável f dependente apenas do parâmetro k,tais que, para qualquer instância (x, k), o algoritmo $\mathcal{A}$ decide se (x, k) é SIM/NÃO em tempo $f(k) \cdot |(x, k)|^c$.

Podemos obter um algoritmo de *kernelização* $\mathcal{K}$ para Q da seguinte forma. Dada uma instância (x, k) de Q, o algoritmo de *kernelização* executa $\mathcal{A}$ em (x, k) por no máximo

$|x|^{c+1}$ passos (onde um *passo* se refere a uma instrução básica de um algoritmo, em sua noção intuitiva). Se em até $|x|^{c+1}$ passos a execução de $\mathcal{A}$ terminar com alguma resposta, o algoritmo $\mathcal{K}$ usa tal resposta para devolver que (x, k) é SIM/NÃO. Se $\mathcal{A}$ não termina após $|I|^{c+1}$ passos executados, devolvemos a própria instância (x, k) como saída do algoritmo de *kernelização* $\mathcal{K}$. Uma vez que $\mathcal{A}$ não terminou em $|x|^{c+1}$ passos, sabemos que $f(k) \cdot |x|^c > |x|^{c+1}$. Portanto, $|x| < f(k)$ e, consequentemente, $|x| + k < f(k) + k$.

Assim, concluímos que o tamanho do *kernel* é limitado superiormente por $f(k) + k$ e, uma vez que $f(k)$ é uma função computável, o limitante superior também será. □

Sabendo do resultado do teorema acima, uma questão natural para qualquer problema sabidamente FPT é se ele admite ou não um *kernel* limitado por uma função polinomial do parâmetro.

Os exemplos estudados de aplicação da técnica de *kernelização* se encontram no Apêndice A.

4 Árvores de busca limitadas

Uma *árvore de busca* é uma representação da execução de um algoritmo recursivo. O algoritmo tenta construir uma solução viável para o problema tomando uma sequência de decisões sobre alguma de suas propriedades. Para cada passo considerado, o algoritmo investiga várias possibilidades para a decisão, *ramificando* em uma série de subproblemas, que são resolvidos separadamente. Uma técnica simples e muito utilizada em complexidade parametrizada envolve garantir que a altura de tal árvore seja, de alguma forma, dependente do parâmetro. Por isso, a chamamos de *árvore de busca limitada* e chamamos tal algoritmo de *ramificação*.

A corretude de um algoritmo de ramificação é justificada argumentando que, no caso de uma instância SIM, alguma sequência de decisões escolhida pelo algoritmo leva a uma solução viável. Se o tamanho total da árvore de busca é limitado por uma função apenas do parâmetro e cada passo leva tempo polinomial, então tal algoritmo de ramificação é FPT.

Um esquema padrão de aplicação da ideia de árvores de busca limitadas no projeto de algoritmos parametrizados é o seguinte. Primeiro, identificamos, em tempo polinomial, um pequeno (constante ou limitado por uma função do parâmetro) subconjunto S de elementos dos quais pelo menos um tem de estar em alguma ou toda solução viável para o problema. Então, resolvemos $|S|$ subproblemas: para cada elemento e de S, criamos um subproblema no qual incluímos e na solução, e resolvemos a tarefa restante com um valor de parâmetro reduzido.

O Apêndice B apresenta aplicações da técnica das árvores de busca limitadas a três problemas parametrizados.

5 Métodos aleatorizados

Em algoritmos aleatorizados, além da entrada, é fornecido também um chamado *fluxo de bits aleatórios*, uma sequência de bits gerada aleatoriamente e que auxiliará de alguma forma o algoritmo na obtenção de uma solução ao problema a ser resolvido.

Chamamos de *algoritmo de Monte Carlo de erro unilateral com falsos negativos* um algoritmo para um problema de decisão que, ao receber uma instância NÃO, sempre devolve NÃO, mas que, ao receber uma instância SIM, tem probabilidade $p \in [0, 1]$ de devolver SIM.

Nesse contexto, uma ideia interessante é buscar formas de fazer com que a probabilidade p seja a maior possível, ao custo de um tempo de execução pior. Dado um algoritmo de Monte Carlo para um problema de decisão, podemos criar um novo algoritmo que repete sua execução c vezes, sendo que esse novo algoritmo devolverá NÃO apenas se o algoritmo de Monte Carlo devolver NÃO em todas as c execuções, o que faz com que ele sempre devolva uma resposta correta para uma instância NÃO. Sendo assim, ele erra a resposta de uma instância SIM apenas se todas as c execuções devolverem NÃO, o que tem probabilidade $(1-p)^c$. Utilizando a desigualdade $1 + x \leq e^x$, obtemos $(1-p)^c \leq (e^{-p})^c = 1/e^{pc}$. Portanto, a probabilidade de devolver a resposta correta para essa instância SIM é pelo menos $1 - 1/e^{pc}$. Dessa forma, para lidarmos bem com instâncias SIM, podemos definir $c = \lceil \frac{1}{p} \rceil$, o que nos dá a probabilidade constante $1 - 1/e$ de devolver a resposta correta.

Assim, se conseguirmos mostrar que a probabilidade p de um algoritmo de Monte Carlo de tempo FPT é igual a $1/(f(k)n^{O(1)})$ para alguma função computável f do parâmetro k do problema, um algoritmo que o repita $f(k)n^{O(1)}$ vezes terá probabilidade de acerto constante para instâncias SIM e seu tempo de execução será $f(k)n^{O(1)}$ vezes maior que o tempo do algoritmo de Monte Carlo, continuando com tempo de execução FPT.

Um exemplo básico de aplicação desse método pode ser encontrado no Apêndice C.1.

5.1 Codificação por cores e separação aleatória

A técnica denominada *codificação por cores* foi proposta em 1995, por Alon, Yuster e Zwick [1], com o intuito de resolver problemas nos quais, dados um grafo H com k vértices e um grafo de entrada G com n vértices, normalmente com n bem maior que k, queremos encontrar um subgrafo de G isomorfo a H.

Por força bruta, podemos resolver o problema em tempo $O(n^k)$, como segue. Primeiro,

fixamos uma sequência arbitrária para os vértices de H, i.e., nomeamos os k elementos de $V(H)$ como $v_1, \ldots, v_k$. Depois, escolhemos uma sequência de k vértices distintos de G e os nomeamos como $u_1, \ldots, u_k$. Por fim, verificamos em tempo polinomial se a vizinhança de cada v_i em H tem o mesmo número de elementos e os mesmos índices da vizinhança de u_i em G. Como o número de maneiras possíveis de formar sequências de k vértices diferentes escolhidos em um conjunto de n vértices é $n!/(n-k)!$, o tempo total será $O(n^k)$.

No entanto, utilizando a técnica de codificação por cores, é possível resolver o problema em tempo de execução $2^{O(k)}n^{O(1)}$, que é FPT quando o parâmetro é k, nos casos em que o grafo H pertence a determinadas classes de grafos, como quado H é uma árvore, por exemplo. A ideia por trás da técnica é colorir aleatoriamente todos os vértices do grafo G com um número de cores escolhido de forma que, se o grafo H existir como subgrafo de G, então há uma alta probabilidade de sua cópia em G ser colorida de modo que possamos encontrá-la eficientemente.

Ao abordarem esta técnica, os autores de [11] citam que, de forma mais geral, é possível obter tempo de execução $2^{O(k)}n^{O(1)}$ não apenas se H for uma árvore, mas quando H tem *treewidth* constante, sendo *treewidth* o assunto abordado na Seção 6.

Um exemplo da aplicação da técnica de codificação por cores pode ser encontrado no Apêndice C.2.

Uma outra técnica, chamada de *separação aleatória*, proposta em 2006 por Cai, Chan e Chan [8], pode ser vista como uma variação da técnica de codificação por cores e é útil quando lidamos com problemas em grafos cujos vértices têm grau limitado. Ela parte da ideia de que se particionarmos aleatoriamente o conjunto de vértices ou de arestas de um grafo em dois conjuntos, a solução buscada poderá ser encontrada eficientemente com certa probabilidade. O Apêndice C.3 traz um exemplo de aplicação da técnica de separação aleatória.

6 *Treewidth*

Nesta seção, abordamos o conceito de *treewidth* de um grafo, que, de maneira informal, é uma medida quantitativa de quão semelhante a uma árvore é a estrutura do grafo. Tal medida surge a partir das chamadas decomposições em árvore, e é uma ferramenta importante em complexidade parametrizada, visto que em grafos com determinados valores de *treewidth* podemos garantir, por exemplo, que alguns problemas sejam tratáveis ou não por parâmetro fixo. Apesar de conceitos equivalentes terem sido introduzidos anterior e independentemente por outros pesquisadores, as definições apresentadas nesta seção foram propostas por Robertson e Seymour, em 1984 [25].

Abordaremos, nas próximas subseções, o que são as decomposições em caminho e em

árvore, bem como as medidas *pathwidth* e *treewidth*, para, na seção seguinte, tratarmos da lógica monádica de segunda ordem ($\mathbf{MSOL_2}$), um formalismo lógico que permite descrever problemas em grafos, e enunciarmos um teorema que permite caracterizar problemas como FPT caso possam ser descritos por $\mathbf{MSOL_2}$.

6.1 Decomposições em caminho e *pathwidth*

Comecemos, então, definindo uma decomposição em caminho.

Definição 1 ([25])**.** Uma *decomposição em caminho* de um grafo G é uma sequência $\mathcal{P} = (X_1, X_2, \ldots, X_r)$ de *bags* $X_i \subseteq V(G)$, para $1 \leq i \leq r$, na qual valem as seguintes propriedades:

(P1) Todo vértice de G está em pelo menos uma *bag*, ou seja, $\bigcup_{i=1}^{r} X_i = V(G)$.

(P2) Para toda aresta uv de G, existe uma *bag* X_ℓ, com $1 \leq \ell \leq r$, que contém ambos os vértices u e v, ou seja, $u, v \in X_\ell$.

(P3) Para todo vértice $u \in V(G)$, se $u \in X_i \cap X_k$ para $i \leq j$, então $u \in X_k$ para todo k tal que $i \leq k \leq j$, ou seja, todas as *bags* no intervalo $X_i \ldots X_j$ contêm u.

Uma decomposição em caminho $\mathcal{P}$ também pode ser vista como um caminho propriamente dito, em vez de uma sequência. Assim, cada uma de suas *bags* serão os vértices (que, para diferenciar dos vértices do grafo, podemos chamar de *nós*) desse caminho, com uma aresta conectando cada par de *bags* consecutivas.

A *largura* de uma decomposição em caminho $\mathcal{P} = (X_1, X_2, \ldots, X_r)$ é definida como a cardinalidade de sua maior *bag* subtraída de uma unidade, ou seja, $\max_{1 \leq i \leq r}(|X_i|) - 1$. A *pathwidth* (*largura em caminho*) de um grafo G, denotada por $\mathrm{pw}(G)$, é a largura mínima possível de uma decomposição em caminho de G. Subtraímos 1 na definição da largura da decomposição em caminho para garantir que a *pathwidth* de um caminho com pelo menos uma aresta seja 1, e não 2.

Como exemplos de *pathwidth* para algumas classes de grafos (com pelo menos uma aresta), podemos citar que um caminho tem *pathwidth* 1, um ciclo tem *pathwidth* 2, uma estrela tem *pathwidth* 1 e uma clique de n vértices tem *pathwidth* $n - 1$.

Uma decomposição em caminho $\mathcal{P} = (X_1, X_2, \ldots, X_r)$ de um grafo G é *boa* se

- $X_1 = X_r = \emptyset$;
- para todo índice i, com $1 \leq i \leq r - 1$, ou existe um vértice $v \notin X_i$ tal que $X_{i+1} = X_i \cup \{v\}$, ou existe um vértice $w \in X_i$ tal que $X_{i+1} = X_i \setminus \{w\}$.

Chamaremos as *bags* da forma $X_{i+1} = X_i \cup \{v\}$ de *bags (ou nós) de introdução*, e as *bags* da forma $X_{i+1} = X_i \setminus \{w\}$ de *bags (ou nós) de esquecimento*. Também diremos que X_{i+1} *introduz* v ou *esquece* w, respectivamente. Note que todo vértice de G é introduzido

e esquecido exatamente uma vez em uma decomposição em caminho boa, o que nos leva a concluir que o número total r de *bags* em uma decomposição em caminho boa é $2|V(G)|+1$.

O lema a seguir nos mostra que qualquer decomposição em caminho pode ser transformada em uma decomposição em caminho boa com, no máximo, a mesma largura.

Lema 3 ([4]). *Se um grafo G admite uma decomposição em caminho de largura no máximo p, então ele também admite uma decomposição em caminho boa de largura no máximo p. Além disso, dada uma decomposição em caminho $\mathcal{P} = (X_1, X_2, \ldots, X_r)$ de G, de largura no máximo p, pode-se, em tempo $O(p^2 \cdot \max(r, |V(G)|))$, calcular uma decomposição em caminho boa de G de largura no máximo p.*

Introduzimos o conceito de decomposições em caminho boas pelo fato de serem muito importantes para resultados baseados em *pathwidth* e algoritmos de programação dinâmica, porém, tais resultados não foram objeto de estudo em nossa pesquisa.

6.2 Decomposições em árvore e *treewidth*

Agora, partimos para a definição de uma decomposição em árvore.

Definição 2 ([25]). Uma *decomposição em árvore* de um grafo G é um par $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$, onde T é uma árvore na qual cada nó t tem atribuído a si uma *bag* $X_t \subseteq V(G)$, tal que as seguintes propriedades valem:

(T1) Todo vértice de G está em pelo menos uma *bag*, ou seja, $\bigcup_{t \in V(T)} X_t = V(G)$.

(T2) Para toda aresta $uv \in E(G)$, existe um nó t de T tal que a *bag* X_t contém ambos os vértices u e v.

(T3) Para todo vértice $u \in V(G)$, o conjunto $T_u = \{t \in V(T) \colon u \in X_t\}$, isto é, o conjunto de nós cujas *bags* contêm u, induz uma subárvore conexa de T.

Note que uma decomposição em caminho é uma decomposição em árvore em que T é um caminho, e assim como citamos para as decomposições em caminho, podemos nos referir aos vértices de T como *nós*, para distingui-los dos vértices do grafo G.

A *largura* de uma decomposição em árvore $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ é igual a $\max_{t \in V(T)}(|X_t|) - 1$, ou seja, o tamanho de sua maior *bag* subtraída de uma unidade. A *treewidth* (*largura em árvore*) de um grafo G, denotada por $\mathrm{tw}(G)$, é a largura mínima possível de uma decomposição em árvore de G. Também subtraímos 1 na definição de *treewidth*, para garantir que a *treewidth* de uma árvore seja 1.

Como exemplos de *treewidth* para algumas classes de grafos (com pelo menos uma aresta), podemos citar que uma árvore (e, consequentemente, um caminho ou uma estrela) tem

treewidth 1, um ciclo tem *treewidth* 2, uma clique de n vértices tem *treewidth* $n-1$, um *grid* $n \times m$ tem *treewidth* $\min\{n, m\}$ e um grafo bipartido completo $K_{n,m}$ tem *treewidth* $\min\{n, m\}$.

Para as decomposições em árvores, também temos o análogo às decomposições em caminho boas e é conveniente pensar em decomposições em árvore boas como árvores enraizadas. Para uma decomposição em árvore $(T, \{X_t\}_{t \in V(T)})$, seja o vértice r a raiz de T. Diremos que uma decomposição em árvore enraizada $(T, \{X_t\}_{t \in V(T)})$ é *boa* se as seguintes condições forem satisfeitas:

- $X_r = \emptyset$ e, para toda folha ℓ de T, $X_\ell = \emptyset$. Ou seja, a raiz e todas as folhas da árvore contêm *bags* vazias.
- Cada um dos nós que não são folhas de T é de um dos três tipos a seguir:
  Nó de introdução: um nó t com exatamente um filho t' tal que $X_t = X_{t'} \cup \{v\}$, para algum vértice $v \notin X_{t'}$ (dizemos que v é *introduzido* em t).
  Nó de esquecimento: um nó t com exatamente um filho t' tal que $X_t = X_{t'} \setminus \{w\}$, para algum vértice $w \in X_{t'}$ (dizemos que w é *esquecido* em t.)
  Nó de junção: um nó t com dois filhos t_1, t_2 tal que $X_t = X_{t_1} = X_{t_2}$.

Por esta definição, diferentemente das decomposições em caminho boas, todo vértice de G é esquecido apenas uma vez, mas pode ser introduzido mais de uma vez.

O lema a seguir é análogo ao Lema 3, agora para decomposições em árvore boas.

Lema 4 ([4]). *Se um grafo G admite uma decomposição em árvore de largura no máximo k, então ele também admite uma decomposição em árvore boa de largura no máximo k. Além disso, dada uma decomposição em árvore $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ de G de largura no máximo k, pode-se, em tempo $O(k^2 \cdot \max(|V(T)|, |V(G)|))$, calcular uma decomposição em árvore boa de G de largura no máximo k e que tem no máximo $O(k \cdot |V(G)|)$ nós.*

Da mesma forma que as decomposições em caminho boas, as decomposições em árvore boas são importantes para os resultados algorítmicos baseados em *treewidth* e em programação dinâmica. Porém, tais decomposições não são conhecidas antecipadamente, então, para os algoritmos de programação dinâmica que as utilizam, assumimos que uma decomposição sempre é fornecida junto com o grafo de entrada. Há formas de encontrar uma decomposição em caminho ou em árvore ótima ou quase ótima para um dado grafo, mas este tópico não foi abordado na pesquisa, apesar de haver um tópico mais avançado no livro estudado em que se exploram algoritmos para computar tais decomposições.

7 Lógica Monádica de Segunda Ordem ($\mathbf{MSOL_2}$)

Esta seção tem como objeto de estudo um formalismo lógico que permite descrever propriedades (e, consequentemente, problemas) em grafos através de fórmulas lógicas. O

formalismo lógico em questão chama-se Lógica Monádica de Segunda Ordem ($\mathbf{MSOL_2}$) e os autores introduzem tal lógica a partir de exemplos.

Dado um grafo G, a seguinte fórmula, $\mathbf{indep}(X)$ verifica se um dado subconjunto de vértices $X \subseteq V(G)$ é independente:

$$\mathbf{indep}(X) = \forall_{u,v \in X} \neg \mathbf{adj}(u, v).$$

Considerando que $\mathbf{adj}(u, v)$ devolve valor verdadeiro se e somente se dois vértices u e v são adjacentes, é fácil observarmos que $\mathbf{indep}(X)$ equivale à condição de independência do conjunto X. A fórmula $\mathbf{indep}(X)$ terá valor verdadeiro se todos os vértices de X forem dois a dois não adjacentes. Se algum par de vértices $u, v \in X$ for adjacente, a fórmula $\neg\mathbf{adj}(u, v)$ devolverá falso e $\mathbf{indep}(X)$ terá valor falso.

$\mathbf{MSOL_2}$ é uma linguagem formal que nos permite expressar propriedades e objetos em grafos, como (subconjuntos de) vértices e arestas. O resultado da aplicação de uma fórmula de $\mathbf{MSOL_2}$ a um grafo é denominado *avaliação* da fórmula no grafo e, para cada grafo em que uma fórmula é avaliada, ela nos devolve um valor *verdadeiro* ou *falso*.

As fórmulas de $\mathbf{MSOL_2}$ possuem *variáveis* que representam diferentes objetos no grafo, podendo haver quatro tipos delas: variáveis para um vértice, variáveis para uma aresta, variáveis para subconjuntos de vértices e variáveis para subconjuntos de arestas. Durante o processo de avaliação da fórmula, toda variável é *avaliada* para algum objeto de tipo apropriado.

Uma fórmula pode receber parâmetros, que são variáveis dadas junto ao grafo, cujas propriedades serão verificadas no grafo, e que são escritas entre parênteses logo após o nome da fórmula. Na fórmula $\mathbf{indep}(X)$, esse parâmetro é o subconjunto X de vértices. Tais parâmetros serão denominados *variáveis livres* da fórmula. Para avaliar adequadamente a fórmula em um grafo, precisamos receber os valores dessas variáveis. Assim, assumimos que o grafo de entrada é *equipado* com os valores de todas as variáveis livres da fórmula $\mathbf{MSOL_2}$ considerada, o que significa que esses valores são fornecidos junto com o grafo. A sequência das variáveis livres de uma fórmula é chamada de *assinatura* sobre a qual a fórmula é escrita.

A partir dos valores das variáveis livres na fórmula, podemos verificar algumas propriedades sobre elas utilizando os operadores lógicos $\wedge$ (conjunção) $\vee$ (disjunção), $\neg$ (negação) e $\implies$ (implicação). No exemplo $\mathbf{indep}(X)$, verificamos se dois vértices u e v são não adjacentes utilizando $\neg\mathbf{adj}(u, v)$, utilizando o operador de negação em $\mathbf{adj}(u, v)$.

O que permite que $\mathbf{MSOL_2}$ expresse as propriedades do grafo são os *quantificadores*, que podem ser o *quantificador universal* $\forall$ ou o *quantificador existencial* $\exists$. Cada quantificador é aplicado a alguma *subfórmula* ψ e introduz uma nova variável sobre a qual itera, para que

essa variável seja usada em ψ. A fórmula $\forall_{v \in V(G)} \psi$, onde ψ é alguma subfórmula que usa a variável de vértice v, pode ser lida como "para todo vértice v no grafo G, ψ é verdadeira". Se ψ é verdadeira para toda avaliação possível de v, então a fórmula inteira $\forall_{v \in V} \psi$ é verdadeira; caso contrário, a fórmula inteira é falsa. A fórmula $\exists_{e \in E(G)} \psi$, onde ψ é alguma subfórmula que usa a variável de aresta e, pode ser lida como "existe (pelo menos) uma aresta e no grafo G tal que ψ é verdadeira". Então, dentre todas as avaliações possíveis da variável e para uma aresta do grafo, se existe pelo menos uma para a qual ψ é verdadeira, então a fórmula inteira $\exists_{e \in E(G)} \psi$ é verdadeira; caso contrário, a fórmula inteira é falsa.

Os exemplos de quantificação foram apenas sobre vértices e arestas individuais, mas também podemos quantificar sobre variáveis de subconjuntos de vértices ($\forall_{X \subseteq V(G)}$ ou $\exists_{X \subseteq V(G)}$), ou variáveis subconjuntos de arestas ($\forall_{C \subseteq E}$ ou $\exists_{C \subseteq E}$). Os operadores lógicos também podem ser usados para combinar fórmulas maiores, como veremos adiante.

Uma informação interessante é que o número 2 subscrito em $\mathbf{MSOL_2}$ significa que é permitida a quantificação sobre subconjuntos de vértices e, também, sobre subconjuntos de arestas. Se permitíssemos apenas a quantificação sobre subconjuntos de vértices, a lógica em questão seria $\mathbf{MSOL_1}$, que é um formalismo lógico com menos poder. As variáveis dos subconjuntos de vértices ou de arestas são chamadas de variáveis *monádicas* e dão nome a esse tipo de lógica.

Agora, podemos ver mais alguns exemplos de fórmulas que podemos escrever utilizando $\mathbf{MSOL_2}$. A fórmula **particao**, contém três variáveis livres de subconjuntos de vértices $X_1, X_2, X_3 \subseteq V(G)$ e verifica se (X_1, X_2, X_3) é um partição de $V(G)$:

$$\begin{aligned}\mathbf{particao}(X_1, X_2, X_3) = \forall_{v \in V(G)} [\, & (v \in X_1 \wedge v \notin X_2 \wedge v \notin X_3) \\ & \vee (v \notin X_1 \wedge v \in X_2 \wedge v \notin X_3) \\ & \vee (v \notin X_1 \wedge v \notin X_2 \wedge v \in X_3) \,].\end{aligned}$$

Utilizando as fórmulas **particao** e **indep**, podemos construir uma fórmula **3colorivel**, que verifica se um grafo G é 3-colorível, isto é, se pode ter seu conjunto de vértices particionado em três partes tais que cada uma delas sejam independentes:

$$\begin{aligned}\mathbf{3colorivel} = \exists_{X_1, X_2, X_3 \subseteq V(G)} [& \mathbf{particao}(X_1, X_2, X_3) \wedge \\ & \mathbf{indep}(X_1) \wedge \mathbf{indep}(X_2) \wedge \mathbf{indep}(X_3)].\end{aligned}$$

Agora, a fórmula $\mathbf{conexoAresta}(C)$, para $C \subseteq E(G)$ verifica se o grafo $G[C]$, subgrafo

de G induzido pelo conjunto de arestas C, é conexo:

$$\textbf{conexoArest}(C) = \forall_{Y \subseteq V} [\, (\exists_{u \in V} u \in Y \wedge \exists_{v \in V} v \notin Y) \\ \implies (\exists_{e \in C} \exists_{u \in Y} \exists_{v \notin Y} \textbf{incide}(u, e) \wedge \textbf{incide}(v, e)) \,].$$

A fórmula $\textbf{grau2}(v, C)$ verifica se o vértice $v \in V(G)$ tem exatamente duas arestas incidentes pertencentes a $C \subseteq E(G)$:

$$\textbf{grau2}(v, C) = \exists_{e_1, e_2 \in C} [\, (e_1 \neq e_2) \wedge \textbf{incide}(v, e_1) \wedge \textbf{incide}(v, e_2) \wedge \\ (\forall_{e_3 \in C} \textbf{incide}(v, e_3) \implies (e_1 = e_3 \vee e_2 = e_3)) \,].$$

Utilizando as duas fórmulas anteriores, podemos construir uma fórmula **hamiltoniano** que verifica se o grafo de entrada é hamiltoniano, isto é, se o grafo possui um ciclo que contém todos os seus vértices. As condições para que o grafo seja hamiltoniano são: (i) existir um subconjunto $C \subseteq E(G)$ que induz um grafo conexo, e (ii) todo vértice de V é adjacente a exatamente duas arestas diferentes de C. Vejamos a fórmula:

$$\textbf{hamiltoniano} = \exists_{C \subseteq E} \textbf{conexoAresta}(C) \wedge \forall_{v \in V} \textbf{grau2}(v, C).$$

7.1 Teorema de Courcelle

Tendo introduzido $\textbf{MSOL}_\textbf{2}$ na subseção anterior, iremos apresentar agora um teorema, proposto por Bruno Courcelle, em 1990 [10], que nos diz que se um problema sobre grafos pode ser descrito por uma fórmula de $\textbf{MSOL}_\textbf{2}$, então tal problema é tratável por parâmetro fixo quando o parâmetro é a *treewidth* do grafo de entrada. Seja φ uma fórmula lógica e seja $\|\varphi\|$ o comprimento da codificação de φ como cadeia (*string*).

Teorema 5 (Teorema de Courcelle [10])**.** *Suponha que φ é uma fórmula de $\textbf{MSOL}_\textbf{2}$ e G é um grafo de n vértices equipado com a avaliação de todas as variáveis livres de φ. Suponha, além disso, que seja fornecida uma decomposição em árvore de G de largura t. Então, existe um algoritmo que verifica se φ é satisfeita em G em tempo $f(\|\varphi\|, t) \cdot n$, para alguma função computável f.*

A demonstração do Teorema de Courcelle não foi abordada em nossa pesquisa.

Como as fórmulas **3colorivel** e **hamiltoniano** da Seção 7 têm comprimentos constantes, ao aplicarmos o Teorema de Courcelle sobre elas obtemos como corolário que os problemas de verificar se um grafo é 3-colorível ou se é hamiltoniano são ambos tratáveis por parâmetro fixo quando parametrizados pela *treewidth*.

Perceba que é necessário que as fórmulas que descrevem os problemas tenham comprimento constante. Se considerarmos o problema Cobertura por Vértices, cujo enunciado pode ser encontrado na Seção A.1, poderíamos expressá-lo com uma fórmula de $\mathbf{MSOL_2}$ quantificando existencialmente k variáveis de vértices, representando os vértices da cobertura e depois verificar se toda aresta de G possui uma das variáveis de vértices quantificadas como um de seus extremos. Porém, o comprimento dessa fórmula dependeria linearmente de k, o que faz com que o Teorema de Courcelle forneça um algoritmo $f(k,t) \cdot n$, e não um algoritmo $f(t) \cdot n$. A existência de um algoritmo $f(k,t) \cdot n$ não é um resultado interessante, pois, como podemos ver na Seção B.1, existe um algoritmo baseado em árvores de busca limitadas para Cobertura por Vértices com tempo $O(n^{O(1)}1,4656^k)$.

Por conta do problema com o comprimento das fórmulas, surge uma variante de otimização do Teorema de Courcelle. Sejam $X_1, X_2, \ldots, X_p$ as variáveis monádicas livres de uma fórmula φ. Em Cobertura por Vértices, teríamos uma variável de subconjunto de vértices X que representa a cobertura por vértices. A fórmula φ verifica se as variáveis livres $X_1, X_2, \ldots, X_p$ satisfazem todas as propriedades solicitadas, como, por exemplo, verificar se toda aresta do grafo tem pelo menos um extremo em X. Então, o problema é encontrar uma avaliação das variáveis $X_1, X_2, \ldots, X_p$ que minimize ou maximize o valor de alguma expressão aritmética $\alpha(|X_1|, |X_2|, \ldots, |X_p|)$, com a condição de $\varphi(X_1, X_2, \ldots, X_p)$ ter valor verdadeiro. Consideremos apenas o caso em que α é uma *função afim*, ou seja, $\alpha(x_1, x_2, \ldots, x_p) = a_0 + \sum_{i=1}^{p} a_i x_i$, para $a_i \in R$.

O teorema a seguir, que pode ser visto como a versão de otimização do Teorema de Courcelle, foi proposto por Arnborg, Lagergren e Seese, em 1991 [2].

Teorema 6 ([2])**.** *Seja φ uma fórmula $\boldsymbol{MSOL_2}$ com p variáveis monádicas livres $X_1, X_2, \ldots, X_p$, e seja $\alpha(x_1, x_2, \ldots, x_p)$ uma função afim. Suponha que temos um grafo G de n vértices e uma decomposição em árvore de G de largura t, e suponha que G esteja equipado com a avaliação de todas as variáveis livres de φ exceto $X_1, X_2, \ldots, X_p$. Então, existe um algoritmo que, em tempo $f(\|\varphi\|, t) \cdot n$, encontra o valor mínimo e máximo de $\alpha(|X_1|, |X_2|, \ldots, |X_p|)$ para os conjuntos $X_1, X_2, \ldots, X_p$ para os quais $\varphi(X_1, X_2, \ldots, X_p)$ é verdadeira, onde f é alguma função computável.*

Como exemplo de aplicação para a versão de otimização do Teorema de Courcelle, podemos escrever a fórmula $\mathbf{cobertura}(X)$, de comprimento constante, verifica se X é uma cobertura por vértices de G:

$$\mathbf{cobertura}(X) = \forall_{e \in E} \exists_{x \in X} \mathbf{incide}(x, e).$$

Aplicando o Teorema 6 nessa fórmula, tomando $\alpha(|X|) = |X|$, concluímos que existe um algoritmo que encontra a cardinalidade mínima de uma cobertura por vértices em tempo $f(t) \cdot n$, para alguma função f.

8 Intratabilidade por parâmetro fixo

Enquanto os outros capítulos estudados do livro tratam de técnicas para demonstrar que existem algoritmos FPT para diversos problemas parametrizados, o capítulo tratado nesta seção aborda como podemos mostrar que alguns problemas parametrizados muito provavelmente não são tratáveis por parâmetro fixo. Assim como a teoria da NP-completude fornece evidências de que vários problemas provavelmente não admitem algoritmos de tempo polinomial, por meio de reduções polinomiais (e assumindo que P $\neq$ NP), parte da teoria da complexidade parametrizada consiste em fornecer evidências, por meio das chamadas reduções parametrizadas, de que alguns problemas parametrizados provavelmente não admitem algoritmos FPT.

É interessante destacarmos alguns comentários feitos pelos autores do livro em relação à importância da aplicação de duas abordagens ao se estudar um problema (parametrizado): a abordagem algorítmica e a abordagem de complexidade. Muitas vezes, as falhas na busca por um algoritmo polinomial (resp. FPT) para um problema pode destacar quais são os casos ou as propriedades que tornam o problema difícil e, assim, nos dar ideias para uma demonstração de intratabilidade polinomial (resp. por parâmetro fixo) para o problema. Da mesma forma, a identificação dos motivos pelos quais uma demonstração de intratabilidade não é bem sucedida pode destacar caminhos para a abordagem algorítmica, ou então quais restrições podem ser aplicadas às instâncias para demonstrar que variações mais restritas do problema são tratáveis de forma eficiente.

Uma outra observação importante é que sempre devemos lembrar que na teoria da NP-completude, consideramos que não há esperança de conseguirmos encontrar algoritmos eficientes para os problemas NP-completos, apesar de não haver provas de que P $\neq$ NP. De forma análoga, na teoria da complexidade parametrizada, partimos da suposição de que não existirão algoritmos FPT para alguns problemas e, a partir de reduções, concluímos o mesmo para vários outros problemas. Porém, no caso da complexidade parametrizada, podemos classificar os problemas intratáveis por parâmetro fixo em uma hierarquia de dificuldade, que abordaremos adiante.

8.1 Reduções parametrizadas

Na teoria da NP-completude, definimos uma redução polinomial como segue.

Definição 3. Sejam $A, B \subseteq \Sigma^*$ dois problemas. Uma *redução (por mapeamento) em tempo polinomial* de A para B é um algoritmo de tempo polinomial que, dada uma instância x do problema A, gera uma instância *equivalente* x' do problema B, isto é, x é uma instância SIM do problema A se e somente se x' é uma instância SIM do problema B.

Note que se existe um algoritmo polinomial para o problema B e uma redução polinomial do problema A para o problema B, então também existe um algoritmo polinomial para A, uma vez que, dada uma instância x de A, podemos executar a redução polinomial, obtendo uma instância x' de B, e resolver x' usando o algoritmo polinomial para B.

Vejamos, agora, a definição de redução para o contexto dos problemas parametrizados, proposta por Downey e Fellows, em 1992 [13].

Definição 4 ([13])**.** Sejam $A, B \subseteq \Sigma^* \times \mathbb{N}$ dois problemas parametrizados. Uma *redução parametrizada* de A para B é um algoritmo que, dada uma instância (x, k) de A, gera uma instância (x', k') de B tal que

(1) (x, k) é *equivalente* a (x', k'), isto é, (x, k) é uma instância SIM de A se e somente se (x', k') é uma instância SIM de B;
(2) $k' \leq g(k)$, para alguma função computável g; e
(3) o tempo de execução de tal algoritmo é $f(k) \cdot |x|^{O(1)}$, para alguma função computável f.

Uma observação sutil e importante sobre a definição acima é sempre que podemos supor que as funções f e g são não decrescentes. Note, também, que o item (2) na definição requer que o parâmetro da nova instância seja limitado superiormente por uma função do parâmetro da instância original.

Do mesmo modo que podemos utilizar as reduções polinomiais para provar a existência de um algoritmo polinomial para um problema A quando existe uma redução dele para um problema B e um algoritmo polinomial para B, podemos provar um resultado análogo para problemas parametrizados.

Teorema 7 ([13])**.** *Sejam $A, B \subseteq \Sigma^* \times \mathbb{N}$ dois problemas parametrizados. Se existe uma redução parametrizada de A para B e B é FPT, então A também é FPT.*

Demonstração. Seja (x, k) uma instância do problema parametrizado A e suponha que existe uma redução parametrizada de A para o problema parametrizado B. A redução parametrizada de A para B fornece uma instância equivalente (x', k') de B em tempo $f(k) \cdot |x|^{c_1}$, com $k' \leq g(k)$ e $|x'| \leq f(k) \cdot |x|^{c_1}$, para uma constante c_1.

Suponha que existe um algoritmo para B com tempo de execução $h(k') \cdot |x'|^{c_2}$, sendo c_2 uma constante e h uma função computável não decrescente. Tal algoritmo resolve (x', k') e, de forma equivalente, resolve (x, k), em tempo no máximo

$$h(k') \cdot |x'|^{c_2} \leq h(g(k)) \cdot (f(k) \cdot |x|^{c_1})^{c_2}.$$

Se somarmos o tempo do algoritmo para B sobre (x', k') com o tempo da redução parametrizada de A para B, o tempo total de execução para resolver (x, k) é no máximo

$$f(k) \cdot |x|^{c_1} + h(g(k)) \cdot (f(k) \cdot |x|^{c_1})^{c_2} \leq f(k) \cdot |x|^{c_1 c_2} + h(g(k)) \cdot f(k)^{c_2} \cdot |x|^{c_1 c_2},$$

que é da forma $f'(k) \cdot |x|^{c_1 c_2}$, onde $f'(k) = f(k) + h(g(k)) \cdot f(k)^{c_2}$ é uma função computável. Portanto, o problema A é tratável por parâmetro fixo. □

Como podemos ver no Apêndice D.6, um dos exercícios propostos foi mostrar que há uma redução de Clique para Conjunto Independente, e vice-versa. Tal redução é uma redução parametrizada, pois o novo parâmetro, em ambas as direções, é o mesmo que o da instância original, i.e., $g(k) = k$. Assim, podemos dizer que os dois problemas são igualmente difíceis no sentido de que um é FPT se e somente se o outro também for. Porém, esses dois problemas são exemplos de problemas para os quais não temos esperança que exista algum algoritmo FPT no caso geral. No Apêndice D.6, mostramos um algoritmo FPT para ambos, mas para um caso restrito, no qual o grafo de entrada é r-regular, para todo valor fixado de r, e o parâmetro é k.

Agora, vejamos um exemplo de cuidado que devemos tomar ao classificar as reduções. Observe que um grafo G de n vértices possui um conjunto independente de tamanho k se e somente G ele possui uma cobertura por vértices de tamanho $n - k$, o que significa que (G, k) é uma instância Sim de Conjunto Independente se e somente se $(G, n-k)$ é uma instância Sim de Cobertura por Vértices. Tal observação já nos dá uma redução em tempo polinomial de Conjunto Independente para Cobertura por Vértices. Porém, essa não é uma redução parametrizada, pois como n pode ser muito maior que k, não existe função $g(k)$ tal que $n-k \leq g(k)$. Não temos esperança que exista uma redução parametrizada de Conjunto Independente para Cobertura por Vértices, pois o segundo é FPT e a existência de tal redução implicaria que Conjunto Independente também é FPT, o que não temos esperança de ser verdade, como já comentado.

A última observação importante é que o item (3) da definição de redução parametrizada nos diz que o tempo de execução de uma redução parametrizada não precisa ser polinomial, podendo ser FPT no parâmetro da instância original, pois a soma do tempo dessa redução

com o tempo de um algoritmo FPT para o problema para o qual um outro é reduzido também será FPT.

8.2 Reduções a partir de Clique

Assumimos que Clique é um problema para o qual muito provavelmente não existe um algoritmo FPT. Esta seção apresenta reduções a partir de Clique como forma de demonstrar que alguns problemas são tão difíceis quanto Clique, de modo que tais problemas só serão FPT se Clique também for. Porém, nem todas as reduções são feitas diretamente a partir de Clique, aproveitando a propriedade transitiva das reduções parametrizadas, como mostra o teorema a seguir.

Teorema 8 (estendido de [13])**.** *Sejam $A, B, C \subseteq \Sigma^* \times \mathbb{N}$ três problemas parametrizados. Se existem reduções parametrizadas de A para B e de B para C, então há uma redução parametrizada de A para C.*

Demonstração. Seja R_1 uma redução parametrizada de A para B, com tempo de execução $f_1(k)\cdot|x|^{c_1}$ e limitante $g_1(k)$ do parâmetro, e seja R_2 uma redução parametrizada de B para C, com tempo de execução $f_2(k)\cdot|x|^{c_2}$ e limitante $g_2(k)$ do parâmetro, assumindo que as funções f_1, f_2, g_1, g_2 são não decrescentes.

A ideia é construir uma redução parametrizada R de A para C utilizando R_1 e R_2. Dada uma instância (x, k) de A, a redução R primeiro executa R_1 para criar uma instância equivalente (x', k') de B, com $|x'| \leq f_1(k)\cdot|x|^{c_1}$, para uma constante c_1, e $k' \leq g_1(k)$. Depois, a partir de (x', k'), R executa a redução R_2 para criar uma instância equivalente (x'', k'') de C, com $|x''| \leq f_2(k')\cdot|x'|^{c_2}$, para uma constante c_2, e $k'' \leq g_2(k')$.

Uma vez que (x, k), (x', k') e (x'', k'') são equivalentes, (x, k) é uma instância Sim de A se e somente se (x'', k'') é uma instância Sim de C. Temos $k'' \leq g_2(k') \leq g_2(g_1(k))$ e, portanto, k'' é limitado por uma função computável de k. O tempo de execução total da redução R é

$$\begin{aligned} f_1(k)\cdot|x|^{c_1} + f_2(k')\cdot|x'|^{c_2} &\leq f_1(k)\cdot|x|^{c_1} + f_2(g_1(k))\cdot(f_1(k)\cdot|x|^{c_1})^{c_2} \\ &\leq (f_1(k) + f_2(g_1(k))\cdot f_1(k))\cdot|x|^{c_1c_2} \\ &\leq f'(k)\cdot|x|^{c_1c_2}, \end{aligned}$$

onde $f'(k) = f_1(k)+f_2(g_1(k))\cdot f_1(k)$ é uma função computável apenas no parâmetro original k. Portanto, a redução R é uma redução parametrizada de A para C. □

A primeira redução que veremos nos mostra que Clique continua difícil quando restringimos que grafo de entrada seja regular, isto é, um grafo em que todos os vértices

possuem o mesmo grau. Note que não estamos definindo um grau específico para os vértices do grafo de entrada como, por exemplo, se disséssemos que o grafo deve ser 3-regular; estamos apenas exigindo que todos eles tenham o mesmo grau, sem fixar qual é esse grau.

Teorema 9 ([6]). *Existe uma redução parametrizada de* Clique *para* Clique *em grafos regulares.*

Demonstração. Dada uma instância (G, k) de Clique, se $k \leq 2$, então o problema é trivial e podemos devolver uma instância Sim ou Não diretamente. Assumimos, então, que $k \geq 3$. Sejam Δ o grau máximo de G e $d_G(v)$ o grau do vértice v em G.

Construiremos um grafo G' como segue, resultando numa instância (G', k):

(1) Iniciando com um grafo G' vazio, adicione Δ cópias distintas $G_1, \ldots, G_\Delta$ de G em G' e seja v_i a cópia de $v \in V(G)$ no grafo G_i.

(2) Para todo vértice $v \in V(G)$, adicione em G' um conjunto V_v de $\Delta - d_G(v)$ novos vértices e adicione arestas entre todo vértice de V_v e todo v_i, para $1 \leq i \leq \Delta$.

Note que, por construção, todos os vértices de G' terão grau Δ e, portanto, G' é um grafo regular.

Agora, temos de provar que G possui uma clique de k vértices se e somente se G' também possui. A ida da implicação é fácil, pois como d cópias de G aparecem como subgrafos em G', sabemos que qualquer clique em G tem uma clique correspondente em G'. Para a volta da implicação, note que os novos vértices, introduzidos no passo (2), não aparecem em nenhum triângulo, pois dois vértices v_i e v_j de G', para qualquer $v \in V(G)$ e para $i \neq j$, não possuem arestas entre si. Portanto, como assumimos que $k \geq 3$, tais vértices não podem fazer parte de uma clique de k vértices. A remoção desses vértices de G' fornece d cópias disjuntas de G e, portanto, qualquer clique de k vértices que aparece nelas implica a existência de uma clique de k vértices em G. □

Sabendo que o complemento de um grafo regular também é regular, obtemos um resultado análogo para Conjunto Independente.

Corolário 10 ([6]). *Existe uma redução parametrizada de* Clique *para* Conjunto Independente *em grafos regulares.*

Nesta redução, o grafo de entrada deve ser regular, mas, novamente, não é especificado o grau de seus vértices. No Apêndice D.6, mostramos que Clique e Conjunto Independente são ambos FPT em grafos r-regulares para todo inteiro fixo r. Tal resultado é interessante pois Conjunto Independente é NP-difícil para grafos 3-regulares.

A próxima redução nos mostra como podemos utilizar a propriedade transitiva das reduções parametrizadas, agora já sabendo que Clique se reduz a Clique em grafos regulares e Conjunto Independente em grafos regulares. Considere o problema Cobertura por Vértices Parcial: a entrada consiste em um grafo G e dois inteiros, k e s, e (G, k, s) é uma instância Sim se G possui um conjunto S de k vértices que cobrem pelo menos s arestas. Dizemos que um conjunto S *cobre* uma aresta xy se pelo menos um de seus extremos x e y pertence em S. Note que Cobertura por Vértices é um caso especial de Cobertura por Vértices Parcial, quando $s = |E(G)|$. Podemos provar o seguinte resultado.

Teorema 11 ([23]). *Existe uma redução parametrizada de* Conjunto Independente *em grafos regulares para* Cobertura por Vértices Parcial *parametrizada por* k.

Demonstração. Seja (G, k) uma instância de Conjunto Independente, onde G é um grafo r-regular. Queremos provar que G tem um conjunto independente de tamanho k se e somente se (G, k, rk) é uma instância Sim de Cobertura por Vértices Parcial.

Para a ida da implicação, se G tem um conjunto independente S de tamanho k, então toda aresta de G é coberta por no máximo um vértice de S. Portanto, os vértices de S, quando somados, cobrem rk arestas.

Para a volta da implicação, uma vez que cada vértice de G cobre exatamente r arestas, só é possível que G tenha um conjunto S de k vértices que cobre rk arestas se nenhuma aresta de G for coberta duas vezes por S. Consequentemente, isso só é possível se os vértices de S formarem um conjunto independente de k vértices. □

Sabendo que Cobertura por Vértices admite um algoritmo FPT, a redução anterior nos mostra que o problema geral Cobertura por Vértices Parcial é provavelmente mais difícil do que Cobertura por Vértices.

No restante desta seção do livro, os autores fornecem várias outras reduções parametrizadas relacionadas a Clique, porém, como passamos superficialmente por este tópico ao final da pesquisa, não nos aprofundamos nos outros resultados e demonstrações, que serviam mais para mostrar particularidades e aplicar algumas observações sobre as reduções parametrizadas.

8.3 A W-hierarquia

Na teoria da NP-dificuldade e NP-completude, muitos dos problemas NP-difíceis possuem reduções polinomiais entre si nas duas direções, como, por definição, os problemas NP-completos, que são todos equivalentes entre si. Porém, para problemas parametrizados intratáveis por parâmetro fixo há categorias diferentes de dificuldade.

Essa observação faz surgir a ideia de uma hierarquia de dificuldade para os problemas que não são FPT. Downey e Fellows [13] introduziram a chamada *W-hierarquia* em uma tentativa de capturar a complexidade exata de vários problemas parametrizados.

Por ser um tema um pouco mais avançado e o último a ser estudado em nossa pesquisa, passamos superficialmente por esta seção do livro, mas é importante ressaltar uma observação dos autores: a W-hierarquia é um tópico um pouco mais profundo e de menor interesse quando queremos apenas mostrar que um dado problema não é FPT. Porém, é de nosso interesse conhecer a terminologia básica do assunto, pois a literatura de complexidade parametrizada muitas vezes passa por seus conceitos.

Vamos a algumas definições.

Definição 5 ([13])**.** Um *circuito booleano* é um grafo orientado acíclico onde os nós são rotulados da seguinte maneira:

- se um nó tem grau de entrada 0, então ele será um *nó de entrada*,
- se um nó tem grau de entrada 1, então ele será um *nó de negação*,
- se um nó tem grau de entrada maior que ou igual a 2, então ele será ou um *E-nó* ou um *OU-nó*.
- existe um único nó com grau de saída 0 que é rotulado como *nó de saída*, sendo também um E-nó ou um OU-nó.

A *profundidade* do circuito é o comprimento máximo de um caminho de um nó de entrada para o nó de saída.

A ideia é atribuir valores 0 ou 1 aos nós de entrada e, seguindo a orientação do circuito (óbvia quando identificamos quais são os nós de entrada ou de saída) e o tipo de cada vértice (negação, E, OU), determinar o valor do nó de saída. Um nó de negação tem valor 0 se recebe uma entrada com valor 1, e vice-versa. Um E-nó recebe valor 1 se todas as suas entradas possuem valor 1; caso contrário, recebe valor 0. Um OU-nó recebe valor 0 se todas as suas entradas possuem valor 0; caso contrário, recebe valor 1. Se o valor do nó de saída é 1 para uma determinada atribuição de valores aos nós de entrada, então dizemos que a atribuição *satisfaz* o circuito. Podemos verificar em tempo polinomial se uma determinada atribuição satisfaz o circuito *booleano*.

Decidir se um circuito *booleano* tem uma atribuição que o satisfaz é um problema NP-completo, sendo 3-SAT seu caso especial. Se definirmos o *peso* de uma atribuição como o número de nós de entrada que recebem o valor 1, podemos enunciar a seguinte versão parametrizada desse problema.

Satisfatibilidade de Circuito Ponderado (WCS)	
Instância:	Um circuito *booleano* C.
Parâmetro:	Um inteiro k.
Questão:	Existe uma atribuição que satisfaz C com peso *exatamente* k?

Note que, para cada valor fixo de k, podemos resolver o problema em tempo polinomial, pois existirão $O(n^k)$ possíveis atribuições de peso exatamente k para C. Porém, o problema provavelmente não é tratável por parâmetro fixo, pois podemos reduzir problemas parametrizados como Conjunto Independente ou Conjunto Dominante (cujas definições são relembradas abaixo) para Satisfatibilidade de Circuito Ponderado.

Conjunto Independente	
Instância:	Um grafo G.
Parâmetro:	Um inteiro k.
Questão:	Existe um conjunto X de k vértices em G tal que $uv \notin E(G)$, para todo $u, v \in X$?

Conjunto Dominante	
Instância:	Um grafo G.
Parâmetro:	Um inteiro k.
Questão:	Existe um conjunto dominante (conjunto $X \subseteq V(G)$ tal que a vizinhança de X é $V(G) \setminus X$) de k vértices em G?

A Figura 1 ilustra a redução para cada um desses problemas.

Finalmente, os níveis da chamada W-hierarquia são definidos restringindo Satisfatibilidade de Circuito Ponderado a várias classes de circuitos. Se $\mathcal{C}$ é uma classe de circuitos, então definimos WCS[$\mathcal{C}$] como a restrição de Satisfatibilidade de Circuito Ponderado onde o circuito de entrada C pertence a $\mathcal{C}$. Para a restrição que nos interessa, precisamos de mais algumas definições. Chamaremos de *nós pequenos* os nós que possuem grau de entrada no máximo 2, e de *nós grandes* os que possuem grau de entrada maior que 2. A *trama* de um circuito é o número máximo de nós grandes em um caminho de um nó de entrada para o nó de saída. Denotamos por $\mathcal{C}_{t,d}$ a classe de circuitos com trama no máximo t e profundidade no máximo d. Podemos, enfim, definir a W-hierarquia.

Definição 6 (W-hierarquia [13])**.** Para $t \geq 1$, um problema parametrizado P pertence à classe W[t] se existir uma redução parametrizada de P para WCS[$\mathcal{C}_{t,d}$] para algum $d \geq 1$.

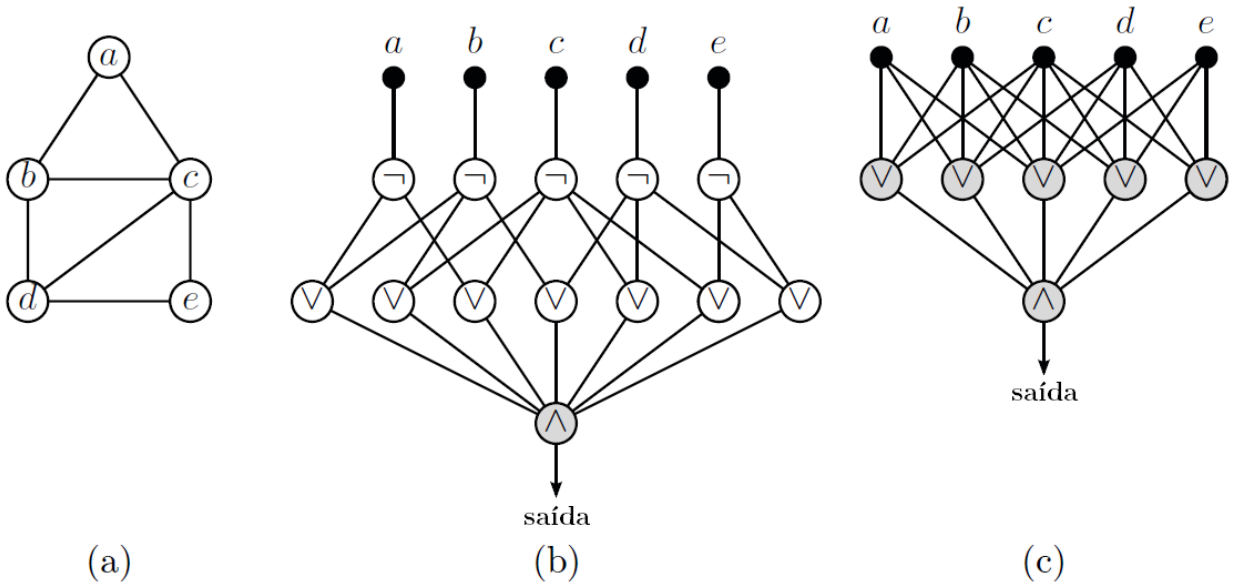

Figura 1: (a) Um grafo G, com 5 vértices e 7 arestas. (b) Um circuito de profundidade 3 e trama 1 satisfeito pelos conjuntos independentes de G. Note que cada OU-vértice corresponde a uma não-aresta (par de vértices não adjacentes) de G e possui grau de entrada 2. (c) Um circuito de profundidade 2 e trama 2 satisfeito pelos conjuntos dominantes de G. Cada OU-vértice corresponde a um vértice de G e seus vizinhos correspondem ao conjunto formado pela união do vértice com seus vizinhos (ou vizinhança fechada do vértice).

Uma observação importante é que a constante 2 na definição de nós pequenos/grandes não é essencial, pois qualquer outra constante maior que 2 resultaria na mesma definição para W[t].

Como comentado e ilustrado na Figura 1, é possível reduzir Conjunto Independente para Satisfatibilidade de Circuito Ponderado para circuitos de profundidade 3 e trama 1. Assim, pela definição de W[t], temos o seguinte resultado.

Proposição 12 ([13])**.** Conjunto Independente *pertence à classe W[1].*

É possível mostrar, analogamente ao que acontece com problemas NP-completos, que todo problema em W[1] pode ser reduzido a Conjunto Independente, isto é, que o problema é W[1]-difícil. A demonstração deste resultado é complexa e não é tratada no livro estudado. Como Conjunto Independente pertence a W[1] e é W[1]-difícil, temos o seguinte resultado.

Teorema 13 ([13])**.** Conjunto Independente *é W[1]-completo.*

Olhando, agora, para Conjunto Dominante, a Figura 1 nos mostra que podemos reduzi-lo para Satisfatibilidade de Circuito Ponderada para circuitos de

profundidade 2 e trama 2. Assim, concluímos que Conjunto Dominante pertence a W[2].

Proposição 14 ([13])**.** Conjunto Dominante *pertence à classe W[2].*

Neste ponto do livro, os autores fazem algumas considerações sobre as classes W[t] para $t \geq 2$. Como nosso objetivo foi abordar superficialmente este tema, só nos atemos ao seguinte resultado, importante para a conclusão da seção.

Teorema 15 ([13])**.** Conjunto Dominante *é W[2]-completo.*

O esperado com a W-hierarquia, como o próprio nome já diz, é que $\mathrm{W}[t] \neq \mathrm{W}[t+1]$, para todo $t \geq 1$. Agora, note que existe o seguinte resultado.

Teorema 16 ([13])**.** *Existe uma redução parametrizada de* Conjunto Independente *para* Conjunto Dominante*.*

Sabendo que Conjunto Independente é W[1]-completo e Conjunto Dominante é W[2]-completo, não há esperança de que exista uma redução de de Conjunto Dominante para Conjunto Independente, pois isso implicaria que Conjunto Independente é simultaneamente W[1] e W[2]-completo. Sob essa suposição, valeria que $\mathrm{W}[1] = \mathrm{W}[2]$, o que é uma contradição com a construção da W-hierarquia.

Como já comentado, a intratabilidade parametrizada é um tópico profundo e vasto dentro desta teoria e nosso objetivo aqui foi apenas ter uma ideia geral de seus fundamentos e suas terminologias.

9 Considerações finais

Ao longo dos doze meses de pesquisa, conseguimos cobrir todos os capítulos planejados no cronograma inicial, com os três primeiros capítulos tratando, respectivamente, das técnicas de *kernelização*, árvores de busca limitadas e métodos aleatorizados, e os dois últimos tratando de tópicos um pouco mais estruturais, que foram *treewidth* e intratabilidade parametrizada.

Com relação às técnicas, conseguimos abordar vários exemplos do livro, como podemos ver nos apêndices deste relatório, complementando várias das demonstrações de lemas e segurança de regras de redução e, também, resolvendo alguns exercícios. Além disso, em cada capítulo que aborda uma técnica, vimos outros resultados auxiliares que permitem aproveitar algumas propriedades das entradas para aplicar a técnica em questão, como a decomposição em coroa, utilizada em kernelização, ou a codificação por cores, utilizada para obter algoritmos FPT aleatorizados, além de alguns lemas importantes em teoria dos grafos.

Os outros dois capítulos traziam alguns resultados que foram apenas enunciados, mas não demonstrados, no livro e em nossa pesquisa, pois tais demonstrações fogem do escopo do livro e, também, de nossos objetivos. Podemos citar como exemplos a demonstração do Teorema de Courcelle ou as demonstrações de que os problemas Conjunto Independente e Conjunto Dominante pertencem, respectivamente, às classes W[1] e W[2]. No entanto, consideramos que a síntese do conteúdo destes dois capítulos contidos no presente relatório, mesmo não trazendo algumas demonstrações, conseguiu abordar de forma geral a utilização e a importância de tais resultados, e o trabalho atingiu o objetivo inicial de introduzir o aluno ao tema, permitindo que as abordagens de (in)tratabilidade por parâmetro fixo possam ser vistas como uma possibilidade quando o aluno estudar problemas computacionais futuramente.

A teoria da complexidade parametrizada cobre vários outros tópicos, e podemos ilustrar isso pelo fato do livro estudado ter muitos outros capítulos mais avançados, tanto sobre técnicas quanto sobre aspectos teóricos. Porém, como citado, conseguimos atingir o objetivo de introduzir o aluno a esta teoria, fazendo com que ele consiga, após este trabalho, ter as técnicas e resultados estudados em sua caixa de ferramentas ao estudar problemas NP-difíceis, bem como para entender melhor a literatura sobre problemas FPT.

Referências

[1] N. Alon, R. Yuster, and U. Zwick. Color-Coding. *Journal of the ACM*, 42(4):844–856, July 1995. doi:10.1145/210332.210337.

[2] S. Arnborg, J. Lagergren, and D. Seese. Easy Problems for Tree-Decomposable Graphs. *Journal of Algorithms*, 12(2):308–340, 1991. ISSN 0196-6774. doi:10.1016/0196-6774(91)90006-K.

[3] A. Becker, R. Bar-Yehuda, and D. Geiger. Randomized Algorithms for the Loop Cutset Problem. *Journal of Artificial Intelligence Research*, 12:219–234, May 2000. doi:10.1613/jair.638.

[4] H. L. Bodlaender and T. Kloks. Efficient and Constructive Algorithms for the Pathwidth and Treewidth of Graphs. *Journal of Algorithms*, 21(2):358–402, 1996. ISSN 0196-6774. doi:10.1006/jagm.1996.0049.

[5] J. F. Buss and J. Goldsmith. Nondeterminism Within P. In C. Choffrut and M. Jantzen, editors, *STACS 91*, pages 348–359, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg. ISBN 978-3-540-47002-1.

[6] L. Cai. Parameterized Complexity of Cardinality Constrained Optimization Problems. *Comput. J.*, 51(1):102–121, jan 2008. ISSN 0010-4620. doi:10.1093/comjnl/bxm086.

[7] L. Cai, J. Chen, R. G. Downey, and M. R. Fellows. Advice Classes of Parameterized Tractability. *Annals of Pure and Applied Logic*, 84(1):119–138, 1997. ISSN 0168-0072. doi:10.1016/S0168-0072(95)00020-8. Asian Logic Conference.

[8] L. Cai, S. M. Chan, and S. O. Chan. Random Separation: A New Method for Solving Fixed-Cardinality Optimization Problems. In H. L. Bodlaender and M. A. Langston, editors, *Parameterized and Exact Computation*, pages 239–250, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-39101-2.

[9] B. Chor, M. Fellows, and D. Juedes. Linear Kernels in Linear Time, or How to Save k Colors in $O(n^2)$ Steps. In J. Hromkovič, M. Nagl, and B. Westfechtel, editors, *Graph-Theoretic Concepts in Computer Science*, pages 257–269, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-30559-0.

[10] B. Courcelle. The Monadic Second-Order Logic of Graphs I: Recognizable Sets of Finite Graphs. *Information and Computation*, 85(1):12–75, 1990. ISSN 0890-5401. doi:10.1016/0890-5401(90)90043-H.

[11] M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*, volume 5. Springer, 2015.

[12] M. Dom, J. Guo, F. Hüffner, R. Niedermeier, and A. Truss. Fixed-Parameter Tractability Results for Feedback Set Problems in Tournaments. *Journal of Discrete Algorithms*, 8 (1):76–86, 2010. ISSN 1570-8667. doi:10.1016/j.jda.2009.08.001.

[13] R. Downey and M. Fellows. Fixed Parameter Tractability and Completeness. *Congressus Numerantium*, 87:191–225, 01 1992.

[14] P. Erdos. Intersection Theorems for Systems of Finite Sets. *Quart. J. Math. Oxford Ser.(2)*, 12:313–320, 1961.

[15] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer Berlin Heidelberg, 2006. doi:10.1007/3-540-29953-x.

[16] Gramm, Niedermeier, and Rossmanith. Fixed-Parameter Algorithms for CLOSEST STRING and Related Problems. *Algorithmica*, 37(1):25–42, Sept. 2003. doi:10.1007/s00453-003-1028-3.

[17] J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Data Reduction and Exact Algorithms for Clique Cover. *ACM J. Exp. Algorithmics*, 13, feb 2009. ISSN 1084-6654. doi:10.1145/1412228.1412236.

[18] P. Hall. On Representatives of Subsets. *Journal of the London Mathematical Society*, s1-10(1):26–30, 01 1935. ISSN 0024-6107. doi:10.1112/jlms/s1-10.37.26.

[19] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973. doi:10.1137/0202019.

[20] D. König. Über Graphen und ihre Anwendung auf Determinantentheorie und Mengenlehre. *Mathematische Annalen*, 77(4):453–465, Dec 1916. ISSN 1432-1807. doi:10.1007/BF01456961.

[21] D. Lokshtanov. *New Methods in Parameterized Algorithms and Complexity*. PhD thesis, University of Bergen, Norway, 2009.

[22] E. M. Luks. Isomorphism of Graphs of Bounded Valence Can Be Tested in Polynomial Time. *Journal of Computer and System Sciences*, 25(1):42–65, 1982. ISSN 0022-0000. doi:10.1016/0022-0000(82)90009-5.

[23] D. Marx. Parameterized Complexity and Approximation Algorithms. *The Computer Journal*, 51(1):60–78, 07 2007. ISSN 0010-4620. doi:10.1093/comjnl/bxm048.

[24] K. Mehlhorn. *Data Structures and Algorithms 2*. Springer Berlin Heidelberg, 1984. doi:10.1007/978-3-642-69897-2.

[25] N. Robertson and P. Seymour. Graph Minors III: Planar Tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984. ISSN 0095-8956. doi:10.1016/0095-8956(84)90013-3.

[26] S. Saurabh. *Exact Algorithms for Optimization and Parameterized Versions of Some Graph Theoretic Problems*. PhD thesis, The Institute of Mathematical Sciences, Chennai, 2009.

[27] M. Sipser. *Introduction to the Theory of Computation*. Cengage Learning, third edition, 2012. ISBN 9781133187790.

A Exemplos de aplicação de *kernelização*

Neste apêndice, apresentamos exemplos de aplicação da técnica de *kernelização*, abordada na Seção 3.

A.1 *Kernel* para Cobertura por Vértices

Dado um grafo G, um conjunto $S \subseteq V(G)$ é denominado uma *cobertura por vértices* se, para toda aresta de G, pelo menos um de seus extremos está em S. Em outras palavras, o grafo $G - S$ não contém arestas e, portanto, $V(G) \setminus S$ é um *conjunto independente*. Dizemos que um vértice pertencente a uma cobertura por vértices *cobre* todas as arestas incidentes a ele.

O primeiro problema para o qual estudamos um resultado de *kernelização* foi Cobertura por Vértices, descrito abaixo, e o algoritmo que veremos foi proposto em [5].

Cobertura por Vértices	
Instância:	Um grafo G.
Parâmetro:	Um inteiro positivo k.
Questão:	O grafo G possui uma cobertura por vértices S tal que $\|S\| \leq k$?

Podemos começar a análise do problema com a observação de que, para uma instância (G, k), se o grafo G possui um vértice v isolado, i.e., de grau 0, então v não pode cobrir nenhuma aresta de G. Portanto, a exclusão de v do grafo G não altera a solução para (G, k). Isso nos leva à primeira regra de redução.

Redução CV.1. Se G contém um vértice v de grau 0, exclua v de G, criando a instância $(G - v, k)$.

Segurança da regra: Se (G, k) é Sim, então existe uma cobertura por vértices S para G com $|S| \leq k$. Se $v \in S$, então v não cobre nenhuma aresta e, certamente, o conjunto $S \setminus \{v\}$ é uma cobertura por vértices de tamanho no máximo k para $G - v$. Consequentemente, $(G - v, k)$ é Sim. Agora, se $v \notin S$, então $S \subseteq V(G - v)$, e também temos que $(G - v, k)$ é Sim. Na direção oposta, se $(G - v, k)$ é Sim, então existe uma cobertura S para $G - v$ com $|S| \leq k$. Certamente, S é uma cobertura também para G, pois v tem grau 0 e, portanto, não precisa participar de nenhuma cobertura para G, por não cobrir nenhuma aresta. Logo, (G, k) também é Sim.

A regra de redução **CV.1** claramente pode ser aplicada em tempo polinomial, bastando armazenar e percorrer uma estrutura que mantenha o grau de cada um dos vértices de G.

Observamos, agora, que se o grafo G contém um vértice v de grau maior que k, então v deve estar, necessariamente, em qualquer cobertura por vértices S de tamanho no máximo k para G. Note que se v não estiver em uma cobertura por vértices, precisamos de pelo menos $k+1$ vértices para cobrir suas arestas adjacentes. Tal observação motiva a seguinte regra de redução.

Redução CV.2. Se G contém um vértice v de grau pelo menos $k+1$, exclua v de G e diminua o parâmetro k em 1 unidade, gerando a instância $(G-v, k-1)$.

Segurança da regra: Se (G,k) é SIM, então existe uma cobertura por vértices S para G, com $|S| \leq k$. Note que S certamente contém v, pois, caso contrário, precisaríamos incluir em S seus pelo menos $k+1$ vizinhos, ultrapassando o valor do parâmetro k. Uma vez que o grafo $G-v$ exclui de G tanto v quanto as arestas adjacentes a v, o conjunto $S \setminus \{v\}$ é uma cobertura de tamanho no máximo $k-1$ para $G-v$, o que significa que $(G-v, k-1)$ é SIM. Na direção oposta, se $(G-v, k-1)$ é SIM, então seja S uma cobertura por vértices para $G-v$, com $|S| \leq k-1$. Note que $S \cup \{v\}$ certamente é uma cobertura por vértices de tamanho no máximo k para o grafo G e, por isso, (G,k) também é SIM.

A regra de redução **CV.2** também pode facilmente ser aplicada em tempo polinomial, bastando, novamente, armazenar e percorrer uma estrutura de dados que armazene os graus dos vértices de G, para identificar um vértice com mais que k vizinhos. Depois, basta criar um novo grafo sem o vértice encontrado e suas arestas adjacentes.

Aplicando exaustivamente as regras **CV.1** e **CV.2**, removeremos de G todos os vértices de grau 0 e de grau maior que k. Observamos, então, que se um grafo tem grau máximo d, então um conjunto de k vértices pode cobrir no máximo kd aresta. O lema a seguir relaciona tal observação ao problema tratado.

Lema 17 ([5])**.** *Se (G,k) é uma instância* SIM *e nenhuma das regras de redução **CV.1** e **CV.2** é aplicável a ela, então $|V(G)| \leq k^2 + k$ e $|E(G)| \leq k^2$.*

Demonstração. Uma vez que a regra **CV.1** não é aplicável, sabemos que o grafo não tem vértices de grau 0. Dessa forma, para toda cobertura por vértices S de G, todo vértice de $G-S$ é necessariamente vizinho de algum vértice de S. Uma vez que a regra **CV.2** não é aplicável, sabemos, também, que todo vértice de G tem grau no máximo k. Assim, temos que $|V(G-S)| \leq k|S|$, e, portanto, $|V(G)| \leq (k+1)|S|$. Como (G,k) é uma instância SIM, existe uma cobertura por vértices S para G tal que $|S| \leq k$, o que implica em $|V(G)| \leq (k+1)k = k^2 + k$. Além disso, como toda aresta de G é coberta por algum vértice de uma cobertura por vértices S e todo vértice pode cobrir no máximo k arestas, concluímos que se $|E(G)| > k^2$, então estamos lidando com uma instância NÃO e, por isso, para uma instância

SIM, vale que $|E(G)| \leq k^2$. □

A partir do Lema 17 surge a seguinte regra de redução, que nos permite limitar o tamanho do *kernel*.

Redução CV.3. Seja (G, k) uma instância tal que as regras **CV.1** e **CV.2** não são aplicáveis a ela. Se $k < 0$ ou $|V(G)| > k^2 + k$ ou $|E(G)| > k^2$ arestas, então concluímos que estamos lidando com uma instância NÃO.

Segurança da regra: A regra é segura pois deriva diretamente do Lema 17. Note que $k < 0$ pode ser obtido aplicando a regra **CV.2** a uma instância $(G, 0)$, e significa que a capacidade do parâmetro já foi ultrapassada, o que faz com que a instância seja NÃO.

A regra **CV.3** também pode ser aplicada em tempo polinomial, pois basta verificar a quantidade de vértices e arestas de G e o valor do parâmetro k. Essa regra nos leva ao resultado principal dessa seção, descrito no seguinte teorema.

Teorema 18 ([5]). *O problema* COBERTURA POR VÉRTICES *admite um kernel com* $O(k^2)$ *vértices e* $O(k^2)$ *arestas.*

A.2 *Kernel* para CONJUNTO DE ARCOS DE RETROALIMENTAÇÃO EM TORNEIOS

Um *torneio* T é uma orientação de um grafo completo, isto é, para todo par de vértices $u, v \in V(T)$, exatamente um entre (u, v) ou (v, u) é uma aresta orientada (ou *arco*) de T. Um conjunto de arcos A de um grafo orientado G é chamado de *conjunto de arcos de retroalimentação (feedback)* se todo ciclo orientado de G contém um arco de A. Em outras palavras, $G - A$ é um grafo orientado acíclico.

O segundo problema para o qual estudamos a técnica de *kernelização* foi CONJUNTO DE ARCOS DE RETROALIMENTAÇÃO EM TORNEIOS, enunciado como segue.

CONJUNTO DE ARCOS DE RETROALIMENTAÇÃO EM TORNEIOS	
Instância:	Um torneio T.
Parâmetro:	Um inteiro não negativo k.
Questão:	O torneio T possui um conjunto de arcos de retroalimentação de tamanho no máximo k?

Quando lidamos com torneios, é importante observar que a remoção de arcos resulta em grafos orientados que deixam de ser torneios. Por isso, no resultado apresentado para este o

problema, será conveniente utilizar a inversão de arcos no lugar da remoção.

O lema a seguir é um resultado bem conhecido e relaciona a existência da chamada *ordenação topológica* para um grafo orientado à existência de ciclos em tal grafo.

Lema 19. *Um grafo orientado G é acíclico se e somente se é possível criar uma rotulação $f\colon V(G) \to \{1, 2, \ldots, |V(G)|\}$ tal que $f(u) < f(v)$ para todo arco $(u, v) \in E(G)$.*

Seja G um grafo orientado e $F \subseteq E(G)$. Além disso, seja $\text{rev}(F) = \{(u, v) : (v, u) \in F\}$. Definimos $G \odot F$ como o grafo orientado obtido de G invertendo todos os arcos do conjunto F, ou seja, $V(G \odot F) = V(G)$ e $E(G \odot F) = (E(G) \setminus F) \cup \text{rev}(F)$.

A partir desta notação, o Lema 19 implica na seguinte observação.

Observação 1. Seja G um grafo orientado e $F \subseteq E(G)$. Se $G \odot F$ é um grafo orientado acíclico, então F é um conjunto de arcos de retroalimentação de G.

A partir da Observação 1, podemos pensar se sua implicação também vale na direção oposta. O lema a seguir nos mostra que sim, mas que é necessário incluir uma condição de minimalidade no conjunto de arcos de retroalimentação envolvido.

Lema 20 ([12])**.** *Dado um grafo orientado G e $F \subseteq E(G)$, o conjunto F é um conjunto de arcos de retroalimentação minimal de G se e somente se F é um conjunto de arcos minimal tal que $G \odot F$ é um grafo orientado acíclico.*

Demonstração. Provaremos, primeiro, a implicação na direção enunciada no lema. Seja F um conjunto de arcos de retroalimentação minimal de G. Para fins de contradição, suponha que $G \odot F$ possui um ciclo orientado C. O ciclo C não pode conter apenas arcos de $E(G) \setminus F$, uma vez que F é um conjunto de arcos de retroalimentação e, por definição, sua remoção de G torna o grafo acíclico. Assim, existem arcos de $\text{rev}(F)$ em C. Sejam $f_1, f_2, \ldots, f_\ell$ os arcos de $E(C) \cap \text{rev}(F)$ na ordem em que aparecem no ciclo C, e seja $\overline{f_i} \in F$ o arco f_i invertido. Pela minimalidade de F, para todo arco $\overline{f_i}$ existe um ciclo orientado C_i em G tal que $F \cap E(C_i) = \{\overline{f_i}\}$. Considere o passeio fechado W em G gerado seguindo o ciclo C, mas toda vez que formos percorrer um arco $f_i \in \text{rev}(F)$, percorremos, em vez disso, o caminho $C_i - \overline{f_i}$. Uma vez que o passeio W é fechado e não contém nenhuma arco de F, obtemos uma contradição com o fato de que F é um conjunto de arcos de retroalimentação de G, uma vez que sua remoção deveria tornar G acíclico. A minimalidade de F vem da Observação 1, que diz que todo conjunto $F \subseteq E(G)$ tal que $G \odot F$ é acíclico também é um conjunto de arcos de retroalimentação de G e, consequentemente, se F não é um conjunto minimal tal que $G \odot F$ é acíclico, F não pode ser um conjunto de arcos de retroalimentação minimal.

No sentido oposto da implicação enunciada no lema, seja F um conjunto de arcos em G tal que $G \odot F$ é um grafo orientado acíclico. Pela Observação 1, F é um conjunto de arcos

de retroalimentação de G e, adicionalmente, é um conjunto de arcos de retroalimentação minimal, pois se um subconjunto $F' \subset F$ é um conjunto de arcos de retroalimentação minimal G, então $G \odot F'$ é um grafo orientado acíclico, uma contradição com a minimalidade de F. □

O teorema a seguir, junto à sua demonstração, fornece um *kernel* para Conjunto de Arcos de Retroalimentação em Torneios.

Teorema 21 ([12])**.** *O problema* Conjunto de Arcos de Retroalimentação em Torneios *admite um kernel com no máximo* $k^2 + 2k$ *vértices.*

Demonstração. Como já observamos, é interessante inverter arcos de um torneio em vez de remover seus arcos, para que o grafo se mantenha sendo um torneio. Assim, pelo Lema 20, um torneio T tem um conjunto de arcos de retroalimentação de tamanho no máximo k se e somente se T se tornar acíclico com a inversão de no máximo k arcos.

Um *triângulo* é um ciclo orientado de comprimento três e, uma vez que um torneio não pode conter dois arcos entre o mesmo par de vértices, um triângulo é o menor ciclo possível em um torneio e, portanto, um torneio acíclico não pode conter triângulos. Isso nos leva à seguinte regra de redução.

Redução CART.1. Se um arco e de T está contido em pelo menos $k+1$ triângulos, então inverta e em T e reduza k em 1 unidade, gerando a instância $(T \odot \{e\}, k-1)$.

Segurança da regra: Se não invertermos e, teremos que inverter pelo menos um arco de cada um dos $k+1$ triângulos que contêm e, o que faz com que e necessariamente pertença a todo conjunto de arcos de retroalimentação de tamanho no máximo k para T.

Da mesma forma que um arco contido em mais que k triângulos tem de ser removido se quisermos eliminar todos os ciclos do torneio com no máximo k inversões de arcos, note que um vértice que não pertence a nenhum triângulo não afeta a solução do problema, pois a inversão de seus arcos adjacentes não muda o número de triângulos do torneio. Essa segunda observação também nos leva a uma regra de redução.

Redução CART.2. Se um vértice v de T não está contido em nenhum triângulo, então exclua v de T, gerando a instância $(T - v, k)$.

Segurança da regra: Dado um arco $e = (x, y)$, chamamos o vértice x de *cauda* de e e o vértice y de *cabeça* de e. Seja $X = N^+(v)$ o conjunto de cabeças dos arcos orientados com cauda v e seja $Y = N^-(v)$ o conjunto de caudas dos arcos orientados com cabeça v. Uma vez que T é um torneio, todos os vértices X e Y formam uma partição de $V(T) \setminus \{v\}$.

Como v não faz parte de nenhum triângulo em T, não existe arco com cauda em X e cabeça em Y. Assim, para qualquer conjunto A de arcos de retroalimentação do torneio $T[X]$ e qualquer conjunto B de arcos de retroalimentação do torneio $T[Y]$, o conjunto $A \cup B$ é um conjunto de arcos de retroalimentação de T. É fácil notar que o inverso também vale, pois, para qualquer conjunto W de arcos de retroalimentação de T, temos que $W \cap E(T[X])$ é um conjunto de arcos de retroalimentação de $T[X]$ e que $W \cap E(T[Y])$ é um conjunto de arcos de retroalimentação de $T[Y]$. Portanto, (T, k) é uma instância Sim se e somente se $(T - v, k)$ é uma instância Sim.

Ambas as regras são aplicáveis em tempo polinomial, bastando verificar, no caso da regra **CART.1**, se os extremos de um arco possuem $k+1$ vizinhos em comum tal que um triângulo é formado com cada um deles e, no caso da regra **CART.2**, se o vértice possui algum par de vizinhos tal que os arcos entre os três formem um triângulo. Além disso, observe que o grafo resultante da aplicação dessas duas regras também é um torneio.

Se (T, k) é instância Sim e nenhuma das regras de redução **CART.1** ou **CART.2** é aplicável a ela, podemos mostrar que T possui no máximo $k^2 + 2k$ vértices, como segue. Seja A um conjunto de arcos de retroalimentação de tamanho no máximo k para T. Para cada um dos no máximo k arcos $e \in A$, note que existem no máximo k vértices que estão em triângulos contendo e, além dos dois extremos de e; portanto, $k + 2$ vértices. Isso acontece pois, caso contrário, a regra **CART.1** seria aplicável. Uma vez que todo triângulo em T possui um dos k arcos de A e todo vértice de T está em um triângulo, concluímos que T tem no máximo $k(k + 2) = k^2 + 2k$ vértices.

Portanto, dada uma instância (T, k), o algoritmo aplica exaustivamente as duas regras de redução, obtendo uma instância equivalente (T', k'). Se o torneio T' possui mais que $k'^2 + k'$ vértices, então o algoritmo devolve que (T, k) é Não. Caso contrário, obtemos o *kernel* enunciado, concluindo a demonstração. □

A.3 *Kernel* para Cobertura de Arestas por Cliques

Dado um grafo G, um conjunto $C \subseteq V(G)$ é uma *clique* se para todo par $u, v \in C$, com $u \neq v$, temos $uv \in E(G)$.

O terceiro problema tratado, chamado Cobertura de Arestas por Cliques, é um exemplo de que nem todos os problemas FPT admitem *kernels* de tamanho polinomial.

Cobertura de Arestas por Cliques	
Instância:	Um grafo G.
Parâmetro:	Um inteiro não negativo k.
Questão:	Existe um conjunto de k subgrafos $H = \{H_1, H_2, \ldots, H_k\}$ de G tal que $E(G) = \bigcup_{i=1}^{k} E(H_i)$ e todo $H_i \in H$ é uma clique?

Em outras palavras, o objetivo do problema é decidir se existem no máximo k cliques em G tais que toda aresta de G pertença a pelo menos uma das cliques. Chamamos o conjunto de cliques H que satisfaz tal propriedade de *cobertura de arestas por cliques* e dizemos que uma clique $H_i \in H$ *cobre* as arestas $E(H_i) \cap E(G)$ em G.

Denotamos por $N(v) = \{u : uv \in E(G)\}$ a *vizinhança* do vértice v em G, e por $N[v] = N(v) \cup \{v\}$ a *vizinhança fechada* de v.

Algumas observações sobre a instância nos levam à criação de regras de redução simples, que foram propostas em [17]. A primeira delas é que um vértice isolado, i.e., de grau 0 não consiste em uma clique, podendo ser removido da instância sem prejuízos.

Redução CAC.1. Se o grafo G contém um vértice isolado v, exclua v de G, gerando a instância $(G - v, k)$.

Segurança da regra: Uma vez que um vértice isolado não é uma clique, tal vértice não estará contido em nenhuma cobertura de arestas por cliques para a instância (G, k) e, consequentemente, uma cobertura de arestas por cliques para (G, k) sempre será uma cobertura de arestas por cliques para $(G - v, k)$, e vice-versa. Logo, (G, k) é uma instância Sim se e somente se $(G - v, k)$ é uma instância Sim.

A segunda observação é que uma aresta isolada é uma clique e, caso exista tal aresta no grafo de entrada, ela só pode ser coberta pela clique que consiste nela mesma.

Redução CAC.2. Se o grafo G contém uma aresta isolada uv, exclua uv de G e diminua k em 1 unidade, gerando a instância $(G - \{u, v\}, k - 1)$.

Segurança da regra: Uma vez que uma aresta isolada uv é uma clique, a única forma de uv pertencer a alguma cobertura de arestas por cliques é se um dos subgrafos da cobertura for a própria aresta uv. Assim, se (G, k) é uma instância Sim, então existe um conjunto de k cliques que cobrem todas as arestas de G em que uv é uma das cliques deste conjunto. Logo, existe um conjunto de $k-1$ cliques que cobrem todas as arestas de $G-\{u, v\}$, sendo portanto $(G - \{u, v\}, k - 1)$ uma instância Sim. Por outro lado, se $(G - \{u, v\}, k - 1)$ é instância Sim, existe cobertura de arestas por cliques com $k - 1$ cliques de $G - \{u, v\}$ que pode ter a clique uv acrescentada para formar uma cobertura de arestas por cliques de tamanho k para G.

Logo, (G, k) também é uma instância SIM.

A próxima observação é que dois vértices u e v tal $N[u] = N[v]$, que são chamados de *gêmeos verdadeiros*, podem ser tratados exatamente da mesma maneira em solução para o problema, o que permite a exclusão de um deles sem prejuízo à solução. Por outro lado, vértices que estejam contidos exatamente no mesmo conjunto de cliques em uma solução são necessariamente gêmeos verdadeiros.

Redução CAC.3. Se o grafo G contém uma aresta uv cujos extremos tenham exatamente a mesma vizinhança fechada, i.e., se $N[u] = N[v]$, então exclua v de G, gerando a instância $(G - v, k)$.

O resultado a seguir limita, então, o tamanho do *kernel* para COBERTURA DE ARESTAS POR CLIQUES. Note que, diferente dos outros exemplos, o tamanho do *kernel* é exponencial, e não polinomial.

Teorema 22 ([17])**.** *O problema* COBERTURA DE ARESTAS POR CLIQUES *admite um kernel com no máximo* 2^k *vértices.*

Demonstração. Primeiramente, provamos a seguinte afirmação.

Afirmação 1. Se (G, k) é uma instância SIM na qual nenhuma das regras de redução **CAC.1-CAC.3** pode ser aplicada, então $|V(G)| \leq 2^k$.

Demonstração. Seja (G, k) uma instância SIM tal que $H = \{H_1, \ldots, H_k\}$ é uma cobertura de arestas por cliques de G. Para fins de contradição, suponhamos que G tem mais que 2^k vértices. Para cada vértice $v \in V(G)$, criamos um vetor b_v de k *bits*, no qual o *bit* i, com $1 \leq i \leq k$, terá valor 1 se e somente se v está contido na clique H_i. Uma vez que a quantidade total de configurações possíveis para um vetor de bits de tamanho k é 2^k e G tem mais do que 2^k vértices, deve haver necessariamente um par de vértices $u, v \in V(G)$, com $u \neq v$, tal que $b_u = b_v$.

Se b_u e b_v são vetores compostos apenas por 0, então a regra **CAC.1** é aplicável. Caso contrário, sabemos que u e v estão contidos nas mesmas cliques. Isso significa que u e v são adjacentes e têm a mesma vizinhança, o que torna possível a aplicação da regra **CAC.2**, no caso de u e v formarem uma aresta isolada, ou da regra **CAC.3**, nos demais casos. Portanto, se G tem mais do que 2^k vértices, pelo menos uma das regras de redução **CAC.1-CAC.3** é aplicável a ele, uma contradição com a afirmação inicial. □

Podemos criar um algoritmo de kernelização com o seguinte funcionamento. Dada uma instância (G, k), o algoritmo aplica as regras **CAC.1-CAC.3** exaustivamente e, caso o grafo

resultante possua mais que 2^k vértices, o algoritmo devolve que a instância é NÃO, utilizando a Afirmação 1. Caso contrário, o algoritmo devolve a instância equivalente reduzida. □

A.4 Decomposição em Coroa

Estudamos, também, uma técnica denominada *decomposição em coroa*, bem utilizada para obter *kernels* para problemas representados em grafos.

Dado um grafo G, chamamos um conjunto $M \subseteq E(G)$ de *emparelhamento* se todo vértice de G incide em no máximo uma aresta de M, ou seja, nenhum par de arestas de M não possui extremos em comum. Para dois subconjuntos de vértices disjuntos $U, W \subset V(G)$, um emparelhamento M é chamado de *emparelhamento de U em W* se toda aresta de M tem um extremo em U e o outro extremo em W e, adicionalmente, todo vértice de U é um extremo de alguma aresta de M. Nesse caso, também dizemos que M *satura* U.

Seguimos com o enunciado da decomposição em coroa.

Definição 7 (Decomposição em coroa [9])**.** Uma *decomposição em coroa* de um grafo G é uma partição de $V(G)$ em três partes C, H e R, tal que:

1. C é um conjunto não vazio.
2. C é um conjunto independente.
3. Não há arestas entre os vértices de C e R.
4. Se E' é o conjunto de arestas entre os vértices de C e H, então E' contém um emparelhamento de tamanho $|H|$, i.e., um emparelhamento de H em C.

A técnica de decomposição em coroa é baseada nos seguintes teoremas clássicos de Kőnig e Hall sobre emparelhamentos, que nos auxiliam a encontrar uma decomposição em coroa em tempo polinomial.

Teorema 23 (Teorema de Kőnig [20])**.** *Em todo grafo bipartido não orientado, o tamanho de um emparelhamento máximo é igual ao tamanho de uma cobertura por vértices mínima.*

Teorema 24 (Teorema de Hall [18])**.** *Seja G um grafo bipartido não orientado com bipartição (V_1, V_2). O grafo G tem um emparelhamento que satura V_1 se e somente se, para todo $X \subseteq V_1$, temos $|N(X)| \geq |X|$.*

Utilizando ambos os resultados anteriores, o seguinte algoritmo, proposto por Hopcroft e Karp, nos permite encontrar um emparelhamento máximo ou uma cobertura por vértices mínima em tempo polinomial.

Teorema 25 (Algoritmo de Hopcroft-Karp [19])**.** *Seja G um grafo não orientado bipartido com n vértices, m arestas e com bipartição (V_1, V_2). Podemos encontrar um emparelhamento*

máximo bem como uma cobertura por vértices mínima de G em tempo $O(m\sqrt{n})$. Além disso, também em tempo $O(m\sqrt{n})$, podemos encontrar um emparelhamento que satura V_1 ou um conjunto minimal $X \subseteq V_1$ tal que $|N(X)| < |X|$.

Por serem resultados conhecidos e que estão presentes aqui para auxiliar na obtenção do resultado para *kernelização*, as demonstrações dos três teoremas anteriores não serão abordadas neste ponto.

O lema a seguir é o resultado principal dessa subseção, utilizando os teoremas anteriores para definir a base da técnica de decomposição em coroa para desenvolver algoritmos de *kernelização*.

Lema 26 (Lema da coroa [9])**.** *Seja G um grafo sem vértices isolados e com pelo menos $3k+1$ vértices. Existe um algoritmo de tempo polinomial que:*

- *encontra um emparelhamento de tamanho $k+1$ em G; ou*
- *encontra uma decomposição em coroa para G.*

Demonstração. Primeiramente, utilizamos um algoritmo guloso para encontrar um emparelhamento maximal M em G, bastando incluir em M arestas de G sem extremos em comum, enquanto isso for possível. Caso $|M| \geq k+1$, então terminamos, pois encontramos um emparelhamento de tamanho $k+1$ em G. Assumimos, então, que $|M| \leq k$, e que V_M é o conjunto dos extremos das arestas de M. Sabemos que $|V_M| \leq 2k$, pois $|M| \leq k$. Uma vez que M é um emparelhamento maximal, o conjunto de vértices $I = V(G) \setminus V_M$ é um conjunto independente.

Considere o grafo bipartido G', formado por todas as arestas de G com um extremo em V_M e outro em I. Utilizando o Algoritmo de Hopcroft-Karp (Teorema 25), podemos encontrar uma cobertura por vértices mínima X e um emparelhamento máximo M' para G' em tempo polinomial. Podemos assumir que $|M'| \leq k$, pois, caso contrário, terminamos. Pelo Teorema de Kőnig (Teorema 23), sabemos que $|X| = |M'|$, e como assumimos que $|M'| \leq k$, temos $|X| \leq k$.

Se nenhum vértice de X está em V_M, então $X \subseteq I$, mas mostraremos que $X = I$. Para fins de contradição, suponha que há um vértice $w \in I \setminus X$. Pelo fato de G não ter vértices isolados, existe uma aresta wz adjacente a w em G'. Por G' ser bipartido, necessariamente $z \in V_M$. Porém, X é uma cobertura por vértices de G' tal que $X \cap V_M = \emptyset$. Isso que implica $w \in X$, uma contradição à suposição de que $w \notin X$, o que prova que só pode ser válido que $X = I$. Com isso, $|I| \leq |X| \leq k$, e G tem no máximo $|I| + |V_M| \leq k + 2k = 3k$ vértices, o que contradiz a afirmação do lema.

Sabendo que $X \cap V_M \neq \emptyset$, podemos obter uma decomposição em coroa (C, H, R) da seguinte forma. Uma vez que $|X| = |M'|$, toda aresta do emparelhamento M' tem exatamente

um extremo em X. Seja M^* o subconjunto de M' tal que toda aresta de M^* tenha exatamente um extremo em $X \cap V_M$ e seja V_{M^*} o conjunto de extremos das arestas em M^*. Para a decomposição em coroa (C, H, R), definimos o conjunto $H = V_M \cap X = V_{M^*} \cap X$, o conjunto $C = V_{M^*} \cap I$, e a parte restante $R = V(G) \setminus (C \cup H) = V(G) \setminus V_{M^*}$. Assim, H é o conjunto de extremos das arestas de M^* que estão em V_M e C é o conjunto de extremos das arestas de M^* que estão em I. O conjunto C é um conjunto independente e, por construção, M^* é um emparelhamento de H em C, que satura H. Além disso, X é uma cobertura por vértices de G', então todo vértice de C pode ser adjacente apenas aos vértices de H e, portanto, H separa C e R, concluindo a prova. □

O Lema 26 da coroa fornece uma propriedade estrutural forte quando o parâmetro limita superiormente o tamanho de uma cobertura por vértices mínima ou de um emparelhamento máximo no grafo de entrada do problema.

Nas duas subseções a seguir, apresentamos dois resultados de *kernelização* que utilizam decomposição em coroa.

A.4.1 Um novo *kernel* para Cobertura por Vértices

Para exemplificar a utilização da decomposição em coroa em *kernelização*, retomamos o primeiro problema estudado, Cobertura por Vértices. Considere uma instância (G, k) em que foi aplicada exaustivamente a regra de redução **CV.1**, descrita na Seção A.1. Com isso, podemos assumir que G não tem vértices isolados. Se $|V(G)| \geq 3k+1$, podemos aplicar o Lema da Coroa (Lema 26) à instância (G, k), obtendo um emparelhamento de tamanho $k+1$ ou uma decomposição em coroa $V(G) = C \cup H \cup R$.

Caso obtenha um emparelhamento de tamanho $k+1$, o algoritmo conclui que (G, k) é uma instância Não, pois não há possibilidade de cobrir todas as arestas de G com no máximo k vértices, uma vez que o G possui $k+1$ arestas sem extremos em comum.

Caso obtenha uma decomposição em coroa, seja M um emparelhamento de H em C. Note que o emparelhamento M nos diz que, para toda cobertura por vértices X do grafo G, X contém pelo menos $|M| = |H|$ vértices de $H \cup C$, que é o número mínimo de vértices necessários para cobrir as arestas de M. Por outro lado, o conjunto H cobre todas as arestas de G que incidem em $H \cup C$. Assim, existe uma cobertura por vértices mínima de G que contém H, e podemos reduzir (G, k) a $(G-H, k-|H|)$. Quando excluímos H de G, os vértices de C ficam isolados e, portanto, serão posteriormente reduzidos pela regra de redução **CV.1**.

Sabendo que na decomposição em coroa teremos $H \neq \emptyset$, sempre será possível reduzir o tamanho da instância (G, k) quando $|V(G)| > 3k$ e G não possuir vértices isolados, o que nos leva ao seguinte resultado.

Teorema 27 ([9]). *O problema* Cobertura por Vértices *admite um kernel com no máximo* $3k$ *vértices.*

A.4.2 *Kernel* para Satisfatibilidade Máxima

Em lógica proposicional, dizemos que uma fórmula lógica F está na forma normal conjuntiva (FNC) se F consiste em uma conjunção ($\wedge$) de cláusulas e cada cláusula consiste em um literal ou uma disjunção ($\vee$) de literais. Por exemplo, a fórmula $F = ((x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge x_4)$ está na forma normal conjuntiva. Estudamos, então, o seguinte problema.

Satisfatibilidade Máxima	
Instância:	Uma fórmula F na forma normal conjuntiva.
Parâmetro:	Um inteiro não negativo k.
Questão:	Existe uma atribuição às variáveis de F que satisfaz pelo menos k cláusulas de F?

Como segunda aplicação da decomposição em coroa, o resultado apresentado nesta seção é o seguinte *kernel* para Satisfatibilidade Máxima.

Teorema 28. *[21] O problema* Satisfatibilidade Máxima *admite um kernel com no máximo* $2k$ *cláusulas e* k *variáveis.*

Demonstração. Seja φ uma fórmula FNC com n variáveis e m cláusulas. Seja ψ uma atribuição arbitrária às variáveis de φ e seja $\neg\psi$ a atribuição obtida pelo complemento da atribuição de ψ, i.e., se ψ atribui valor lógico verdadeiro a alguma variável x, então $\neg\psi$ atribui valor lógico falso a x, e vice-versa. Note que ou ψ ou $\neg\psi$ satisfaz pelo menos $m/2$ cláusulas, uma vez que toda cláusula é satisfeita por pelo menos uma das duas atribuições. Dessa forma, se $m \geq 2k$, então (φ, k) é uma instância Sim, pois certamente teremos pelo menos k cláusulas satisfeitas.

Primeiro fornecemos um kernel com $n < k$ variáveis. Seja G_φ um grafo bipartido com bipartição (X, Y), onde X é o conjunto das variáveis de φ e Y é o conjunto de cláusulas de φ. Existe uma aresta entre uma variável $x \in X$ e uma cláusula $c \in Y$ em G_φ se e somente se ou x ou $\neg x$ estiver em c. Com isso, se existir um emparelhamento de X em Y em G_φ, então existe uma atribuição de verdade que satisfaz pelo menos $|X|$ cláusulas, pois podemos definir o valor de cada variável em X de tal forma que a cláusula emparelhada a ela seja satisfeita. Isso implica que pelo menos $|X|$ cláusulas são satisfeitas. Caso $k \leq |X|$, então (φ, k) é uma instância Sim, pois satisfazemos pelo menos k cláusulas. Caso contrário, $k > |X| = n$, e obtemos o kernel desejado.

Agora, se φ tem pelo menos $n \geq k$ variáveis, podemos, em tempo polinomial, ou reduzir φ a uma instância equivalente de tamanho menor ou encontrar uma atribuição para as variáveis que satisfaça pelo menos k cláusulas, concluindo que a instância é SIM.

Suponha que φ tem pelo menos k variáveis. Utilizando o Teorema de Hall (Teorema 24) e o Algoritmo de Hopcroft-Karp (Teorema 25), é possível encontrar em tempo polinomial ou um emparelhamento de X em Y ou um conjunto minimal $C \subseteq X$ tal que $|N(C)| < |C|$. No caso de encontrar um emparelhamento, sabemos que a instância é SIM, e terminamos. Então, suponha que encontramos um conjunto minimal C com a propriedade descrita. Sejam $H = N(C)$ e $R = V(G_\varphi) \setminus (C \cup H)$. Note que não existe arestas entre os vértices de C e R, e $G[C]$ é um conjunto independente. Escolhemos algum vértice $x \in C$ arbitrário. Existe um emparelhamento de $C \setminus \{x\}$ em H, uma vez que $|N(C')| \geq |C'|$ para todo $C' \subseteq C \setminus \{x\}$. Como $|C| > |H|$, temos que o emparelhamento de $C \setminus \{x\}$ em H é um emparelhamento de H em C. Portanto, (C, H, R) forma uma decomposição em coroa de G_φ.

Provaremos que todas as cláusulas em H são satisfeitas em toda atribuição que satisfaça o número máximo de cláusulas. Suponha que ψ seja uma atribuição que satisfaça o número máximo de cláusulas, mas que não satisfaça todas as cláusulas em H. Fixamos uma variável $x \in C$. Seja ψ' a atribuição obtida de ψ da seguinte forma: para toda variável y em $C \setminus \{x\}$, definimos o valor lógico de y de modo que a cláusula em H emparelhada a y seja satisfeita. Uma vez que $N(C) \subseteq H$ e ψ' satisfaz todas as cláusulas em H, a atribuição ψ' satisfaz mais cláusulas que ψ, o que contradiz a suposição de que ψ é uma atribuição que satisfaz o número máximo de cláusulas.

Portanto, uma instância (φ, k) é SIM se e somente se $(\varphi \setminus H, k - |H|)$ também é SIM, permitindo a seguinte regra de redução.

Redução SM.1. Sejam (φ, k) e H como descritos na demonstração. Remova H de φ e diminua k em $|H|$ unidades, gerando a instância $(\varphi \setminus H, k - |H|)$.

A regra de redução **SM.1** pode ser aplicada em tempo polinomial, pois tanto o Lema da Coroa quanto o Algoritmo de Hopcroft-Karp nos garantem tempo polinomial. Se aplicarmos exaustivamente a regra **SM.1**, a instância resultante (φ', k'), caso não possa ser identificada como SIM por alguma verificação descrita anteriormente, terá no máximo $2k'$ cláusulas e no máximo k' variáveis, o que conclui a demonstração do teorema. □

A.5 Lema do Girassol

Outro resultado clássico estudado, com aplicações em *kernelização*, é o chamado *Lema do Girassol* ou *Lema de Erdős-Rado*.

Um *girassol* com k *pétalas* e um *núcleo* Y é uma coleção de conjuntos $S_1, \ldots, S_k$ tais que $S_i \cap S_j = Y$ para todo $i \neq j$, com $1 \leq i, j \leq k$. Os conjuntos $S_i \setminus Y$ são pétalas e nenhum deles pode ser vazio. No entanto, o núcleo Y pode ser vazio, e, consequentemente, uma família de conjuntos disjuntos também é um girassol.

Teorema 29 (Lema do Girassol ou Lema de Erdős-Rado [14])**.** *Seja $\mathcal{F}$ uma família de conjuntos (sem duplicatas) sobre um universo U, tal que cada conjunto em $\mathcal{F}$ tem exatamente d elementos. Se $|\mathcal{F}| > d!(k-1)^d$, então $\mathcal{F}$ contém um girassol com k pétalas e tal girassol pode ser computado em tempo polinomial em $|\mathcal{F}|$, $|U|$ e k.*

Demonstração. Provamos por indução em d. Para $d = 1$, a família $\mathcal{F}$ é formada apenas por conjuntos de um único elemento e a afirmação vale, pois, uma vez que não há duplicatas em $\mathcal{F}$, a família é formada por conjuntos disjuntos. Assim, temos $|\mathcal{F}| > 1!(k-1)^1 = k-1$, o que implica em $|\mathcal{F}| \geq k$, i.e., a família $\mathcal{F}$ sempre terá um girassol com k pétalas e núcleo vazio.

Seja $d \geq 2$ e suponha que $\mathcal{F}$ é uma família de conjuntos de cardinalidade no máximo d sobre um universo U, tal que $|\mathcal{F}| > d!(k-1)^d$.

Seja $\mathcal{A} = \{A_1, \ldots, A_t\} \subseteq \mathcal{F}$ uma família maximal de t conjuntos disjuntos dois a dois em $\mathcal{F}$. Se $t \geq k$, então $\mathcal{A}$ é um girassol com pelo menos k pétalas (e núcleo vazio).

Assumimos, então, que $t < k$ e seja $X = \bigcup_{i=1}^{t} A_i$. Então $|X| \leq d(k-1)$, uma vez que cada um de seus $t \leq k-1$ conjuntos tem no máximo d elementos. Como $\mathcal{A}$ é maximal, todo conjunto de $\mathcal{F}$ tem interseção não vazia com pelo menos um conjunto de $\mathcal{A}$, i.e., $F \cap X \neq \emptyset$. Caso contrário, haveria um conjunto em $\mathcal{F}$ disjunto de todos os conjuntos de $\mathcal{A}$, contradizendo sua maximalidade. Portanto, pelo Princípio da Casa dos Pombos, existe um elemento $u \in X$ contido em pelo menos

$$\frac{|\mathcal{F}|}{|X|} > \frac{d!(k-1)^d}{d(k-1)} = (d-1)!(k-1)^{d-1}$$

conjuntos de $\mathcal{F}$.

Identificamos, então, todos os conjuntos de $\mathcal{F}$ contendo o elemento u, e construímos uma família $\mathcal{F}'$ de conjuntos de cardinalidade exatamente $d-1$ removendo u de cada conjunto de $\mathcal{F}$ que o contenha. Sabemos que $|\mathcal{F}'| > (d-1)!(k-1)^{d-1}$, então, pela hipótese de indução, $\mathcal{F}'$ contém um girassol $\{A'_1, \ldots, A'_k\}$ com k pétalas. Portanto, $\{A'_1 \cup \{u\}, \ldots, A'_k \cup \{u\}\}$ é um girassol em $\mathcal{F}$ com k pétalas, concluindo a prova.

No que diz respeito ao tempo de execução necessário para computar um girassol, a demonstração do lema pode ser transformada em um algoritmo de tempo polinomial, como segue. Primeiro, escolhemos, de forma gulosa, uma família maximal de conjuntos disjuntos dois a dois. Depois, verificamos se o tamanho da família escolhida é pelo menos k,

devolvendo a própria família como girassol, em caso positivo. Caso contrário, encontramos um elemento u contido no número máximo de conjuntos em $\mathcal{F}$, e fazemos uma chamada recursiva em conjuntos de cardinalidade $d-1$, obtidos da remoção do elemento u dos conjuntos que o contenham. □

Na subseção a seguir, veremos como o Lema do Girassol pode ser utilizado para construir um algoritmo de *kernelização* para um problema específico.

A.5.1 *Kernel* para d-Hitting Set

Como uma aplicação do Lema do Girassol (Teorema 29), veremos como obter um *kernel* para o problema d-Hitting Set.

d-**Hitting Set**	
Instância:	Uma família $\mathcal{A}$ de conjuntos sobre um universo U, onde cada conjunto na família tem cardinalidade no máximo d.
Parâmetro:	Um inteiro positivo k.
Questão:	Existe um subconjunto $H \subseteq U$, de tamanho no máximo k, que contém pelo menos um elemento de cada conjunto em $\mathcal{A}$?

Um conjunto que contém pelo menos um elemento de cada conjunto de uma família $\mathcal{A}$, como o conjunto H procurado no problema, será denominado um *hitting set* de $\mathcal{A}$.

Teorema 30 ([15])**.** *O problema d-Hitting Set admite um kernel com no máximo $d! \cdot k^d$ conjuntos e no máximo $d! \cdot k^d \cdot d^2$ elementos.*

Demonstração. Note que se $\mathcal{A}$ contém um girassol $S = \{S_1, \ldots, S_{k+1}\}$, então todo *hitting set* de $\mathcal{A}$ de no máximo k elementos tem interseção com o núcleo Y de S. Suponha, para fins de contradição, que H é um *hitting set* de $\mathcal{A}$ tal que $|H| \leq k$ e H não tem interseção com Y. Assim, H certamente tem interseção com cada uma das $k+1$ pétalas $S_i \setminus Y$, uma vez que H contém pelo menos um elemento de cada conjunto S_i de S. Porém, isso contradiz o fato de que $|H| \leq k$.

Essa observação nos leva à seguinte regra de redução.

Redução HS.1. Seja $(U, \mathcal{A}, k)$ uma instância de d-Hitting Set tal que $\mathcal{A}$ contém um girassol $S = \{S_1, \ldots, S_{k+1}\}$ com núcleo Y. A nova instância será $(U', \mathcal{A}', k)$, onde $\mathcal{A}' = (\mathcal{A} \setminus S) \cup \{Y\}$ é a família obtida excluindo de $\mathcal{A}$ todos os conjuntos $\{S_1, \ldots, S_{k+1}\}$ e adicionando um novo conjunto Y, e $U' = \bigcup_{X \in \mathcal{A}'} X$.

Observe que a regra de redução exclui o girassol S de $\mathcal{A}$, mas, logo em seguida, adiciona seu núcleo Y à família novamente. Assim, a regra não exclui de $\mathcal{A}$ todos os elementos contidos nos conjuntos de S, mas apenas os elementos de suas pétalas, que certamente não participam de nenhum *hitting set* de $\mathcal{A}$. Tendo removido elementos que nunca participarão de algum *hitting set*, a instância $(U, \mathcal{A}, k)$ é SIM se e somente se $(U', \mathcal{A}', k)$ é SIM.

Podemos, então, criar um algoritmo de *kernelização* com o seguinte funcionamento. Se, para algum valor $d' \in \{1, \ldots, d\}$ a quantidade de conjuntos em $\mathcal{A}$ de tamanho exatamente d' for maior que $d'! \cdot k^{d'}$, então o algoritmo aplica o Lema do Girassol (Teorema 29) para encontrar um girassol de tamanho $k+1$ e, em seguida, aplica a regra de redução **HS.1** em tal girassol. Aplicando esses dois passos exaustivamente, o algoritmo obterá uma família $\mathcal{A}'$ de tamanho no máximo $d! \cdot k^d \cdot d$. Se em algum momento o algoritmo encontrar um girassol com núcleo vazio, então ele devolve que a instância é NÃO, pois isso significa que não existe *hitting set* de $\mathcal{A}'$ de tamanho no máximo k. Caso contrário, como todo conjunto de $\mathcal{A}'$ contém no máximo d elementos, o número de elementos no *kernel* é no máximo $d! \cdot k^d \cdot d^2$.

Como o Lema do Girassol nos garante tempo polinomial para encontrar um girassol em uma família de conjuntos, caso exista, e a remoção dos elementos das pétalas de um girassol também pode ser feito em tempo polinomial, o algoritmo de *kernelização* leva tempo polinomial em $|U|$, $|\mathcal{A}|$ e k. □

B Exemplos de aplicação das árvores de busca limitadas

Neste apêndice, apresentamos exemplos de aplicação da técnica das árvores de busca limitadas, tratada na Seção 4.

B.1 Árvores de busca em Cobertura por Vértices

Uma estratégia recursiva simples para Cobertura por Vértices seria, dados um grafo G e um inteiro k, escolher uma aresta uv arbitrariamente e resolver dois problemas recursivamente: $(G - u, k - 1)$ ou $(G - v, k - 1)$. A ideia é que se um dentre u ou v está na solução, o outro não precisa estar. Claramente, como casos base temos o caso em que $k = 0$ e G ainda possui arestas, de forma que não há solução possível, e o caso em que G não possui arestas, de forma que os vértices já escolhidos formam uma solução para o problema.

Observe que, no pior caso, o algoritmo sempre faz duas chamadas recursivas, diminuindo seu parâmetro em uma unidade a cada nível. Por isso, a árvore de busca terá no máximo $O(2^k)$ folhas, o que nos mostra que esse algoritmo é de parâmetro fixo para o problema Cobertura por Vértices [24]. Porém, é possível obter um algoritmo mais eficiente ao analisarmos melhor a estrutura do problema.

Considere um grafo G qualquer e seja $\Delta(G)$ o grau máximo de G. Se $\Delta(G) = 1$, então certamente G contém apenas arestas e/ou vértices isolados, de forma que qualquer cobertura por vértices ótima terá exatamente um extremo de cada aresta.

Suponha, então, que $\Delta(G) \geq 2$ e seja v um vértice de grau $d(v) = \Delta(G)$. Note que se v não faz parte da cobertura por vértices, então todos os seus vizinhos farão. Assim, em vez de remover um dos dois extremos de uma aresta em cada chamada recursiva, podemos remover um vértice, reduzindo k em 1 unidade, ou remover todos os seus vizinhos, reduzindo k em pelo menos $d(v) \geq 2$ unidades.

Assim, para cada nó interno da árvore de busca cujo parâmetro é i existem duas subárvores: uma para o mesmo algoritmo executado com o parâmetro $i - 1$, e uma chamada recursiva com parâmetro no máximo $i - 2$. Assim, o tempo de execução de cada nó interno (ou da raiz) com parâmetro i é dado pela fórmula recursiva $T(i) \leq T(i-1) + T(i-2)$, caso $i \geq 2$, e $T(i) = 1$, caso contrário. Com isso, o número de folhas de nossa árvore de busca é limitado por $T(k)$. Resolvendo a recorrência, temos

$$\begin{aligned} T(k) &= T(k-1) + T(k-2) \leq 1,6181^{k-1} + 1,6181^{k-2} \\ &= 1,6181^{k-2}(1,6181 + 1) \leq 1,6181^{k-2}(1,6181)^2 \leq 1,6181^k. \end{aligned}$$

Portanto, chegamos ao seguinte resultado.

Teorema 31 (adaptado de [24])**.** Cobertura por Vértices *pode ser resolvido em tempo* $O(n^{O(1)}1,6181^k)$.

De fato, é possível melhorar o resultado ainda mais, conforme veremos em breve. Uma primeira observação importante é mostrada no teorema a seguir.

Teorema 32. Cobertura por Vértices *pode ser resolvido em tempo polinomial se* $\Delta(G) \leq 2$.

Demonstração. Grafos com grau máximo no máximo 2 são a união de ciclos, caminhos e vértices isolados, de forma que uma cobertura por vértices para tais grafos pode ser obtida a partir da união das coberturas de vértices de suas componentes, que são facilmente encontradas. □

No caso anterior, o algoritmo fazia apenas chamadas recursivas com vértices de grau pelo menos 2. Agora, considere que estas chamadas ocorrerão apenas para vértices com grau pelo menos 3. Com isso, a relação de recorrência para o número de vértices da árvore de busca passa a ser $T(k) \leq T(k-1) + T(k-3)$, para $k \geq 3$, e $T(k) = 1$ nos demais casos. Usando a mesma técnica anterior para resolver a recorrência, é possível demonstrar o seguinte resultado.

Teorema 33. Cobertura por Vértices *pode ser resolvido em tempo* $O(n^{O(1)}1,4656^k)$.

Sabendo que o resultado obtido foi melhor quando limitamos o grau máximo do grafo para 2, é natural questionar se uma ideia semelhante poderia ser aplicada em grafos de grau máximo igual a 3. Porém, o fato de Cobertura por Vértices ser NP-difícil para grafos com essa propriedade torna tal ideia inviável.

B.2 Árvores de busca em Conjunto de Vértices de Retroalimentação

Dado um grafo G, dizemos que um conjunto $R \subseteq V(G)$ é um *conjunto de vértices de retroalimentação* de G se $G - R$ é um grafo acíclico (ou seja, uma *floresta*). Veremos, a seguir, um algoritmo de tempo $k^{O(k)} \cdot n^{O(1)}$, proposto em [26], para o seguinte problema.

Conjunto de Vértices de Retroalimentação	
Instância:	Um grafo G.
Parâmetro:	Um inteiro não negativo k.
Questão:	Existe um conjunto de vértices de retroalimentação de tamanho no máximo k em G?

Pela forma como o algoritmo funciona, precisaremos tratar G como um *multigrafo*, i.e., um grafo que pode conter *arestas múltiplas* (mais de uma aresta entre o mesmo par de vértices) e *laços* (uma aresta com os dois extremos no mesmo vértice). Por convenção, um laço contribui com 2 unidades para o grau de seu vértice, e tanto uma aresta dupla (duas arestas entre o mesmo par de vértices) quanto um laço são considerados ciclos.

A etapa inicial do algoritmo consiste em aplicar as seguintes regras de redução, com a ordem de prioridade dada pela ordem em que são apresentadas.

A primeira regra surge da observação de que todo vértice que possua um laço necessariamente estará contido em um conjunto de vértices de retroalimentação R de G.

Redução CVR.1. Se houver um laço em um vértice v, exclua v do grafo e diminua k em 1 unidade, criando a instância $(G - v, k - 1)$.

Segurança da regra: Se a instância (G, k) é SIM, então existe um conjunto de vértices de retroalimentação R em G tal que $|R| \leq k$. Sabemos que $G - R$ é um grafo acíclico e que um laço é considerado um ciclo. Portanto, concluímos que v certamente está contido em R, uma vez que seu laço é um ciclo que só pode ser removido do grafo com a exclusão de v. Além disso, $R \setminus \{v\}$ é um conjunto de vértices de retroalimentação de tamanho $k - 1$ em $G - v$ e, por isso, a instância $(G - v, k - 1)$ é SIM. Na direção oposta, se $(G - v, k - 1)$ é uma instância SIM, então existe um conjunto de vértices de retroalimentação R' em $G - v$ tal que $|R'| \leq k - 1$. Note que $R' \cup \{v\}$ é um conjunto de vértices de retroalimentação em G, tanto para remover o laço em v do grafo quanto para remover outros possíveis ciclos que passem por v em G. Como $|R' \cup \{v\}| \leq k$, segue que (G, k) também é uma instância SIM.

A segunda regra surge da observação de que cópias excedentes de arestas múltiplas não influenciam o conjunto de soluções viáveis para (G, k).

Redução CVR.2. Se houver uma aresta e de multiplicidade maior que 2, reduza sua multiplicidade para 2, criando a instância (G', k), onde G' é definido como segue. Seja $\{e_1, e_2, \ldots, e_\ell\}$ o conjunto das $\ell > 2$ cópias de e em G, definimos $V(G') = V(G)$ e $E(G') = E(G) \setminus \{e_3, \ldots, e_\ell\}$.

Segurança da regra: Se a instância (G, k) é SIM, então existe um conjunto de vértices de retroalimentação R em G tal que $|R| \leq k$. Sabemos que $G - R$ é um grafo acíclico e que arestas múltiplas são ciclos. Portanto, os extremos de e, digamos u e v, estão contidos em pelo menos um ciclo em G. No grafo G', os vértices u e v continuam contidos em um ciclo entre si em G', e cada um continuará nos outros mesmos ciclos com os demais vértices, se existirem tais ciclos, o que faz com que conjunto R não seja afetado. Assim, R será um conjunto de vértices de retroalimentação de k vértices em G' e a instância (G', k) é SIM.

Na direção oposta, se (G', k) é uma instância SIM, então existe um conjunto de vértices de retroalimentação R' em G' tal que $|R'| \leq k$. Como arestas múltiplas são ciclos, os extremos de e, digamos u' e v', participam de pelo menos um ciclo em G'. Assim, os vértices u' e v' continuarão contidos em pelo menos um ciclo entre si em G, e cada um continuará nos outros mesmos ciclos com os demais vértices, se existirem tais ciclos. Assim, R' é um conjunto de vértices de retroalimentação de no máximo k vértices em G e a instância (G, k) é SIM.

A terceira regra surge da observação de que um vértice de grau 0 ou 1 certamente não participa de nenhum ciclo em G, podendo ser excluído do grafo sem prejuízos à solução.

Redução CVR.3. Se houver um vértice v de grau no máximo 1 em G, exclua v do grafo, criando a instância $(G - v, k)$.

Segurança da regra: Se a instância (G, k) é SIM, então existe um conjunto de vértices de retroalimentação R em G tal que $|R| \leq k$. Se $v \notin R$, então tome $R' = R$. Se $v \in R$, tome $R' = R \setminus \{v\}$. Note que, em ambos os casos, R' certamente é um conjunto de vértices de retroalimentação de no máximo k vértices no grafo $G - v$. Assim, a instância $(G - v, k)$ é SIM. Na direção oposta, se $(G - v, k)$ é uma instância SIM, existe um conjunto de vértices de retroalimentação R' em $G - v$ tal que $|R'| \leq k$. Como v tem grau no máximo 1 em G, certamente v não participa de ciclos em G e, portanto, o conjunto R' é um conjunto de vértices de retroalimentação de tamanho no máximo k também em G. Logo, (G, k) também é uma instância SIM.

A quarta regra surge da observação de que, tendo aplicado exaustivamente as regras **CVR.1-CVR.3** no grafo, um vértice de grau exatamente 2 é visto pela solução como equivalente aos seus vizinhos. Assim, em vez de incluir tal vértice na solução, podemos incluir um de seus vizinhos.

Redução CVR.4. Se houver um vértice v de grau 2 em G, exclua v do grafo e conecte seus dois vizinhos por uma nova aresta, criando a instância (G', k), onde G' é definido como segue. Sejam u e w os dois vizinhos de v em G, definimos $V(G') = V(G) \setminus \{v\}$ e $E(G') = (E(G) \setminus \{uv, wv\}) \cup \{uw\}$.

Observações sobre a regra: O vértice v não pode ter um laço, pois, caso o tivesse, a regra de redução **CVR.1** seria aplicável a v, o que garante que u e w sejam distintos de v. Além disso, é possível que o vértice v incida apenas em uma aresta dupla, i.e., $u = w$, caso em que excluímos v do grafo e adicionamos um laço ao vizinho único de v. Nesse último caso, sabemos que a regra **CVR.1** será aplicada no vizinho único de v logo após a aplicação da regra **CVR.4**.

Segurança da regra: Se a instância (G, k) é SIM, então existe um conjunto de vértices de retroalimentação R em G tal que $|R| \leq k$. Podemos considerar três casos. No primeiro caso, v não participa de ciclos. Assim, se $v \in R$, então tome $R' = R \setminus \{v\}$, e se $v \notin R$, então tome $R' = R$. No segundo caso, v participa de pelo menos um ciclo e possui dois vizinhos distintos u e w, então ambas as suas arestas incidentes uv e vw participam de ciclos. Assim, se $v \in R$, tome $R' = R \setminus \{v\} \cup \{u\}$ ou $R' = R \setminus \{v\} \cup \{w\}$, e se $v \notin R$, tome $R' = R$. No terceiro caso, v participa de pelo menos um ciclo e tem apenas um vizinho u $(= w)$, adjacente a ele por uma aresta uv de multiplicidade 2, então v participa de exatamente um ciclo. Assim, se $v \in R$, tome $R' = R \setminus \{v\} \cup \{u\}$, e se $v \notin R$, tome $R' = R$ e note que $u \in R$. Nesse caso, ao excluir v do grafo incluiremos um laço em u, e a regra de redução **CVR.1** garantirá que u, agora com laço, estará na solução do grafo G', corroborando a suposição de que, entre v e u, podemos manter u no conjunto R'. Note que, em todos os casos, R' é um conjunto de vértices de retroalimentação em G' tal que $|R'| \leq k$. Portanto, em qualquer um dos casos, a instância (G', k) é SIM.

Por outro lado, se (G', k) é uma instância SIM, existe um conjunto de vértices de retroalimentação R' em G' tal que $|R'| \leq k$. Suponhamos que $u \neq w$ e que nenhum deles faz parte de R'. Isso significa que em G', u e w não fazem parte de ciclos. Nesse caso, note que v também não faz parte de ciclos em G, já que u e w são seus únicos vizinhos. Agora, se pelo menos um dos vértices u e w estiver em R', significa que sua remoção torna G' acíclico. Note que em $G - R$, o grau de v será 0 ou 1 e, portanto, v não pode estar em ciclos. Agora, suponhamos que $u = w$, i.e., a aresta uw é um laço. Assim, certamente $u \in R'$. Em G, teremos o ciclo formado pela aresta uv, de multiplicidade 2. Portanto, Se removermos R' de G, o grafo será acíclico, pois a remoção de u de G deixará v com grau 0 em G, e certamente v não participará de nenhum conjunto de vértices de retroalimentação para G. Em todos os casos, R' é um conjunto de vértices de retroalimentação em G. Portanto, a instância (G, k) também é SIM.

Sobre o tempo de execução, vamos observar que todas as regras podem ser aplicadas em tempo polinomial. Para aplicar **CVR.1** e **CVR.2**, é fácil identificar vértices com laços e arestas de multiplicidade maior que 2 através de uma matriz de adjacências do grafo que contenha a informação de quantas arestas existem entre cada par de vértices, já que podemos ter arestas múltiplas. Para aplicar **CVR.3** e **CVR.4**, é fácil identificar o grau de cada um dos vértices se guardarmos essa informação na construção do grafo, por exemplo, em um vetor de inteiros.

Deste ponto em diante, podemos assumir que as regras de redução **CVR.1-CVR.4** já foram aplicadas exaustivamente ao grafo G da instância (G, k), com nenhuma delas sendo

mais aplicável ao grafo. Assim, sabemos que G possui grau mínimo pelo menos 3. Precisamos, também, adicionar a seguinte regra, que interrompe o algoritmo se já tivermos excedido a capacidade do parâmetro.

Redução CVR.5. Se $k < 0$, o algoritmo termina e concluímos que (G, k) é uma instância NÃO.

Segurança da regra: Entre as quatro regras de redução anteriores, a regra **CVR.1** é única que diminui o valor de k. Se, em algum ponto da execução do algoritmo, tivermos $k = 0$ e a regra **CVR.1** for aplicada, a instância gerada por sua aplicação terá $k = -1$. Isso significa que foi identificado em $(G, 0)$ um vértice que certamente faz parte da solução, por ter um laço (que é um ciclo), mas como tínhamos $k = 0$, a única maneira de a instância ser SIM seria com o grafo sendo acíclico. Portanto, quando obtemos um valor negativo para k, a instância correspondente só pode ser NÃO.

Agora, partiremos para a descrição do algoritmo de ramificação para o problema, cuja essência está nas seguintes observações informais. Se R é um conjunto de vértices de retroalimentação de G, então, por definição, $G - R$ é uma floresta. Sabemos que uma floresta de n vértices tem no máximo $n - 1$ arestas. Assim, $G - R$ tem no máximo $|V(G)| - |R| - 1$ arestas. Veremos adiantes que $G - R$ não pode ter muitos vértices de grau alto, de forma que ao escolher $f(k)$ vértices de graus mais altos no grafo, toda solução de tamanho no máximo k tem que conter um desses vértices de grau alto.

Seja $(v_1, v_2, \ldots, v_n)$ uma ordenação não crescente de $V(G)$ pelos graus dos vértices, isto é, $d(v_1) \geq d(v_2) \geq \cdots \geq d(v_n)$. Seja $V_{3k} = \{v_1, \ldots, v_{3k}\}$. O lema a seguir é a base para o algoritmo proposto.

Lema 34. *[[26]] Todo conjunto de vértices de retroalimentação de tamanho no máximo k em G contém pelo menos um vértice de V_{3k}.*

Demonstração. Começamos provando a seguinte afirmação.

Afirmação 2. Para todo conjunto de vértices de retroalimentação R de G,

$$\sum_{v \in R} (d(v) - 1) \geq |E(G)| - |V(G)| + 1.$$

Demonstração. O grafo $F = G - R$ é uma floresta e, por isso, sabemos que $|E(F)| \leq |V(G)| - |R| - 1$. Sabendo, também, que toda aresta de $E(G) \setminus E(F)$ incide em um vértice de R, teremos

$$\sum_{v \in R} d(v) + |V(G)| - |R| - 1 \geq |E(G)|.$$

Uma vez que

$$\sum_{v \in R} d(v) - |R| = \sum_{v \in R} (d(v) - 1),$$

teremos

$$\sum_{v \in R} (d(v) - 1) + |V(G)| - 1 \geq |E(G)| \implies \sum_{v \in R} (d(v) - 1) \geq |E(G)| - |V(G)| + 1.$$

Isso conclui a prova. □

Suponhamos, para fins de contradição, que existe um conjunto de vértices de retroalimentação R de tamanho no máximo k tal que $R \cap V_{3k} = \emptyset$. Pela construção de V_{3k}, para todo $v \in R$, o grau de v é no máximo o mínimo dos graus dos vértices de V_{3k}. Sabendo que $k \geq |R|$, temos, portanto

$$\sum_{i=1}^{3k} (d(v_i) - 1) \geq 3 \left(\sum_{v \in R} (d(v) - 1) \right).$$

Utilizando a Afirmação 2, temos

$$3 \left(\sum_{v \in R} (d(v) - 1) \right) \geq 3(|E(G)| - |V(G)| + 1).$$

Como $R \subseteq V(G) \setminus V_{3k}$, temos

$$\sum_{i=3k+1}^{n} (d(v_i) - 1) \geq \sum_{v \in R} (d(v) - 1) \geq (|E(G)| - |V(G)| + 1).$$

Portanto,

$$\sum_{i=1}^{n} (d(v_i) - 1) \geq 4 \cdot (|E(G)| - |V(G)| + 1).$$

No entanto, toda aresta é contada duas vezes quando somamos os graus do grafo, uma

vez para cada um de seus extremos. Ou seja, $\sum_{i=1}^{n} d(v_i) = 2|E(G)|$. Assim, obtemos

$$\begin{aligned}
&\sum_{i=1}^{n}(d(v_i) - 1) = 2|E(G)| - |V(G)| \\
&\implies 4(|E(G)| - |V(G)| + 1) \leq 2|E(G)| - |V(G)| \\
&\implies 4|E(G)| - 4|V(G)| + 4 \leq 2|E(G)| - |V(G)| \\
&\implies 4|E(G)| - 2|E(G)| + 4 \leq 4|V(G)| - |V(G)| \\
&\implies 2|E(G)| + 4 \leq 3|V(G)|.
\end{aligned}$$

Porém, a última inequação é uma contradição com o fato de todo vértice de G ter grau pelo menos 3, que implica em $2|E(G)| \geq 3|V(G)|$. □

Finalmente, utilizamos o Lema 34 para obter um algoritmo para Conjunto de Vértices de Retroalimentação.

Teorema 35 ([26]). *Existe um algoritmo para* Conjunto de Vértices de Retroalimentação *que é executado em tempo* $(3k)^k n^{O(1)}$.

Demonstração. Dado um grafo G e um inteiro não negativo k, o algoritmo primeiro aplica as regras de redução **CVR.1-CVR.5** exaustivamente, na ordem dada. Quando não pudermos mais aplicar nenhuma das regras, ou teremos concluído que a instância é Não, ou obteremos uma instância reduzida (G', k') tal que G' tem grau mínimo pelo menos 3 e $k' \leq k$.

Se G' é vazio, então concluímos que estamos lidando com uma instância Sim, pois $k' \geq 0$ e um conjunto vazio é uma solução viável para o problema. Caso contrário, seja $V_{3k'}$ o conjunto de $3k'$ vértices de maiores graus em G'. Pelo Lema 34, qualquer solução R para (G', k') contém pelo menos um vértice de $V_{3k'}$.

Dessa forma, o algoritmo ramifica a escolha de um desses vértices, isto é, para todo vértice $v \in V_{3k'}$, há uma recursão com a instância $(G' - v, k' - 1)$. Se um dos ramos devolver uma solução R' de tamanho $k' - 1$ para $G' - v$, então certamente $R' \cup \{v\}$ é um conjunto de vértices de retroalimentação de tamanho no máximo k' para G'. Caso contrário, devolvemos que a instância fornecida é Não.

Observamos, por fim, que a altura da árvore de busca não excede k', pois, a cada chamada recursiva, diminuímos o parâmetro em 1 unidade. Como toda etapa ramifica em no máximo $3k'$ subproblemas e a altura não excede k', o número de nós na árvore de busca não excede $(3k')^{k'} \leq (3k)^k$. Cada subproblema pode ser resolvido em tempo polinomial, uma vez que as regras de redução **CVR.1-CVR.5** são aplicáveis em tempo polinomial e verificar se um grafo é vazio obviamente também é. Como o número de nós na árvore é no máximo $(3k)^k$ e cada

nó é resolvido em tempo polinomial, o tempo total do algoritmo será $(3k)^k n^{O(1)}$, concluindo a prova. □

B.3 Árvores de busca em Cadeia Mais Próxima

A última aplicação estudada das árvores de busca foi para o seguinte problema.

Cadeia Mais Próxima	
Instância:	Uma sequência $X = (x_1, \ldots, x_k)$ de k cadeias, descritas sobre um alfabeto Σ e de comprimento L.
Parâmetro:	Um inteiro não negativo d.
Questão:	Decidir se existe uma cadeia y de comprimento L sobre Σ tal que $d_H(y, x_i) \leq d$ para todo $i \in \{1, \ldots, k\}$, onde $d_H(s, t)$ é a *distância de Hamming* entre as cadeias s e t, ou seja, o número de posições em que s e t diferem.

Chamamos a cadeia y da questão do problema de *cadeia central*, e note que o problema é parametrizado por d, que representa a distância máxima permitida entre a cadeia central e as cadeias de entrada.

Denotaremos a letra na p-ésima posição de x por $x[p]$. Assim, $x = x[1]x[2]\cdots x[L]$ para uma cadeia de comprimento L. Diremos que as cadeias x e y *diferem* na p-ésima posição se $x[p] \neq y[p]$.

Seja uma instância (X, d). Representaremos essas cadeias como uma matriz de caracteres de dimensão $k \times L$. Assim, a j-ésima coluna do conjunto é a sequência de caracteres $x_1[j], x_2[j], \ldots, x_k[j]$.

Uma coluna será classificada como *ruim* se contiver pelo menos dois símbolos diferentes. Caso contrário, a coluna será classificada como *boa*. Assim, é fácil escolher a j-ésima letra da solução y, caso a j-ésima coluna da entrada seja boa, pois basta tomar $y[j] = x_1[j] = x_2[j] = \cdots = x_k[j]$. Essa observação nos leva à seguinte regra de redução.

Redução CMP.1. Exclua todas as colunas boas da matriz.

Segurança da regra: Seja (X, d) uma instância qualquer e suponha que a j-ésima coluna de X é boa. Construa a instância (X', d) tal que X' é o conjunto obtido excluindo a j-ésima coluna de X. Suponha que (X, d) é Sim. Então existe uma cadeia y que difere de cada uma das k cadeias de X em no máximo d posições. Note que $y[j] = x_1[j] = x_2[j] = \cdots = x_k[j]$. Então, a cadeia y' obtida de y excluindo-se a j-ésima letra de y irá diferir de cada uma das k

cadeias de X' também em no máximo d posições. Logo, (X', d) é SIM. Por outro lado, se (X', d) é SIM. Então existe uma cadeia y' que difere de cada uma das k cadeias de X' em no máximo d posições. Construa uma cadeia y obtida incluindo uma nova letra em y', de forma que essa letra ocupe a j-ésima posição de y e seja igual a $x_1[j] = x_2[j] = \cdots = x_k[j]$. Por construção, y difere de cada uma das k cadeias de X em no máximo d posições. Logo, (X, d) é SIM. Note que essa explicação pode ser facilmente estendida para o caso em que há mais de uma coluna boa.

A regra **CMP.1** pode ser aplicada em tempo linear $O(kL)$ se, na construção da matriz, coletarmos a informação sobre a quantidade de símbolos em cada coluna.

O lema a seguir nos diz que, caso a instância seja SIM, o número de colunas ruins na matriz é limitado.

Lema 36 ([16])**.** *Para toda instância* SIM *de* CADEIA MAIS PRÓXIMA*, a matriz correspondente de dimensão $k \times L$ contém no máximo kd colunas ruins.*

Demonstração. Primeiro, fixamos uma cadeia central y. Para toda coluna ruim j existe uma cadeia $x_{i(j)}$ tal que $x_{i(j)}[j] \neq y[j]$, pois, caso contrário, a coluna teria sido classificada como boa e excluída pela regra **CMP.1**. Sabendo que toda cadeia x_i difere da cadeia central y em no máximo d posições, então, para cada i, temos $i(j) = i$ em no máximo d posições j. Portanto, existem no máximo kd colunas ruins na matriz. □

A partir do Lema 36, surge uma nova regra de redução, que identifica algumas instâncias NÃO. Não será necessário provar a segurança da regra, uma vez que ela deriva diretamente do lema.

Redução CMP.2. Se houver mais de kd colunas ruins na matriz, concluímos que a instância é NÃO.

A regra **CMP.2** é aplicável em tempo linear $O(kd)$ bastando percorrer a matriz para verificar quais são as colunas ruins.

A ideia do algoritmo de ramificação proposto é a seguinte. Fixamos uma cadeia central y e escolhemos uma das cadeias fornecidas, digamos x_1, como uma cadeia candidata, que podemos chamar de z. Enquanto houver uma cadeia x_i, com $i \in \{1, \ldots, k\}$, tal que $d_H(x_i, z) \geq d + 1$, então haverá pelo menos uma posição p onde z e x_i diferem na qual $y[p] = x_i[p]$. Dessa forma, podemos ramificar $d + 1$ formas de aproximar a cadeia candidata z de y; aproximar, nesse contexto, significa que escolhemos uma posição p na qual a cadeia candidata z e x_i diferem e definimos $z[p] := x_i[p]$. O número de possibilidades será limitado,

uma vez que a cada passo ficamos mais perto de y e, inicialmente, temos $z = x_1$ e $d_H(x_1, y) \leq d$.

O teorema a seguir contém o resultado principal estudado para esse problema e sua demonstração formaliza a ideia acima.

Teorema 37 ([16])**.** *O problema* Cadeia Mais Próxima *pode ser resolvido em tempo* $O(kL + kd(d+1)^d)$.

Demonstração. O algoritmo começa aplicando as regras de redução **CMP.1** e **CMP.2**. Se elas não forem mais aplicáveis, teremos k cadeias $x_1, x_2, \ldots, x_k$ de comprimento $L \leq kd$ cada.

Definimos, então, um novo problema equivalente. Dada uma cadeia candidata z e um inteiro $d' \leq d$, queremos verificar se existe uma cadeia central y para as cadeias $x_1, x_2, \ldots, x_k$ tal que $d_H(y, z) \leq d'$. Esse problema é equivalente a instância de Cadeia Mais Próxima quando $z = x_1$ e $d' = d$.

Vamos resolver esse problema de maneira recursiva. Se z é uma cadeia central, então devolvemos z. Caso contrário, se $d' = 0$, então devolvemos que não existe tal cadeia central y. Se não cairmos em nenhum dos casos, então sabemos que $d' > 0$ e que existe uma cadeia x_i com $d_H(x_i, z) > d$.

Chamaremos de $\mathcal{P}$ um conjunto de $d+1$ posições arbitrárias nas quais x_i e z diferem, uma vez que sabemos que elas diferem em mais de d posições. Para qualquer cadeia central y, temos $y[p] = x_i[p]$ para pelo menos uma posição $p \in \mathcal{P}$. Portanto, é possível ramificar em $|\mathcal{P}| = d + 1$ subproblemas: para toda posição $p \in \mathcal{P}$, definimos $z_p = z$ exceto na posição p onde $z_p[p] = x_i[p]$, e resolvemos recursivamente o novo problema para $(z_p, d' - 1)$.

A corretude desta ramificação está na observação de que se existe uma cadeia central y com $d_H(z, y) \leq d'$, então, para uma posição $p \in \mathcal{P}$ em que $x_i[p] = y[p]$, vale que $d_H(z_p, y) \leq d_H(z, y) - 1 \leq d' - 1$.

Agora, com relação ao tempo de execução, note que a árvore de busca construída pelo algoritmo tem altura no máximo d, e todo nó tem no máximo $d+1$ filhos. Consequentemente, a quantidade de nós da árvore de busca é $O((d+1)^d)$. Cada etapa do algoritmo pode ser implementada em tempo linear $O(kd)$ como segue. Podemos construir, inicialmente, uma tabela contendo a distância entre a primeira cadeia candidata e todas as outras cadeias da entrada. Verificar a distância entre a cadeia candidata e todas as demais é a etapa mais custosa, com tempo $O(kd)$, e precisamos gastar esse tempo cada vez que ramificamos em um novo subproblema, para atualizar a tabela de cada chamada recursiva. Assim, o tempo total da etapa de ramificação inteira será $O(kd(d+1)^d)$. Somado ao tempo de aplicar, antes de tudo, as regras de redução **CMP.1** e **CMP.2**, que é $O(kL)$, chegamos ao tempo de execução total do algoritmo, $O(kL + kd(d+1)^d)$, o que completa a prova do teorema. □

C Exemplos de aplicação de métodos aleatorizados

Nesta seção, trazemos exemplos de aplicação de métodos aleatorizados para tratar problemas parametrizados, como as técnicas abordadas na Seção 5.

C.1 Algoritmo aleatorizado para Conjunto de Vértices de Retroalimentação

Aqui, retomamos o problema Conjunto de Vértices de Retroalimentação, abordado na Seção B.2, mas agora aplicando a ele a técnica de aleatorização [3].

Primeiramente, observamos que usaremos as regras de redução e observações feitas na Seção B.2, lidando com *multigrafos*. Assim, tendo aplicado exaustivamente as regras de redução **CVR.1-CVR.5**, o grafo resultante G não contém laços, contém apenas arestas simples ou duplas e tem grau mínimo 3. O lema a seguir mostra que, em um grafo com essas características, a maioria das arestas tem extremos em vértices da solução ao problema.

Lema 38 ([3])**.** *Seja G um multigrafo com n vértices, com grau mínimo pelo menos 3. Para todo conjunto de vértices de retroalimentação R de G, mais da metade das arestas de G tem pelo menos um extremo em R.*

Demonstração. Seja $H = G - R$. Toda aresta em $E(G) \setminus E(H)$ é incidente a um vértice em R, uma vez que a remoção de R de G para gerar o grafo H exclui tanto os vértices de R quanto as arestas incidentes a eles. Assim, a afirmação do lema equivale a dizer que $|E(G) \setminus E(H)| > |E(H)|$. Entretanto, como H é uma floresta, sabemos que $|V(H)| > |E(H)|$, sendo suficiente mostrar que $|E(G) \setminus E(H)| > |V(H)|$.

Seja J o conjunto de arestas com um extremo em R e o outro em $V(H)$ e sejam $V_{\leq 1}$, V_2 e $V_{\geq 3}$ os conjuntos de vértices em $V(H)$ que tenham, respectivamente, grau no máximo 1, exatamente 2, e pelo menos 3 na floresta H. Como G tem grau mínimo pelo menos 3, cada vértice em $V_{\leq 1}$ contribui com pelo menos duas arestas distintas para J e cada vértice em V_2 contribui com pelo menos uma aresta para J (em ambos os casos, sabemos que a diferença entre o grau de um vértice em G e o grau do mesmo vértice em H, se houver, se deve ao fato de existirem vizinhos dele em G que estão em R). Pelo fato de H ser uma floresta, temos também que o número de folhas é maior que o número de vértices internos e, portanto, em particular, $|V_{\leq 1}| > |V_{\geq 3}|$.

Podemos juntar todas essas informações para mostrar que $|E(G) \setminus E(H)| > |V(H)|$,

provando o lema:

$$|E(G) \setminus E(H)| \geq |J| \geq 2|V_{\leq 1}| + |V_2|$$
$$\implies |E(G) \setminus E(H)| > |V_{\leq 1}| + |V_2| + |V_{\geq 3}| = |V(H)|.$$

Isso conclui a demonstração do lema. □

A partir do Lema 38, tendo aplicado exaustivamente as regras de redução **CVR.1-CVR.5** ao grafo de entrada, se escolhermos uma aresta uniformemente ao acaso, estaríamos escolhendo com probabilidade $1/2$ uma aresta com pelo menos um de seus extremos na solução. Se, em seguida, escolhermos independentemente ao acaso um dos extremos da aresta, estaríamos escolhendo um vértice da solução com probabilidade pelo menos $1/4$. Repetindo esse processo k vezes, obtemos um algoritmo que resolve Conjunto de Vértices de Retroalimentação em tempo polinomial, com probabilidade pelo menos 4^{-k}.

Teorema 39 ([3])**.** *Existe um algoritmo aleatorizado de tempo polinomial que, dada uma instância (G, k) de* Conjunto de Vértices de Retroalimentação*, ou reporta uma falha ou encontra um conjunto de vértices de retroalimentação em G de tamanho no máximo k. Além disso, se o algoritmo receber uma instância* Sim*, ele devolverá uma solução com probabilidade pelo menos 4^{-k}.*

Demonstração. Suponha que aplicamos exaustivamente as regras de redução **CVR.1-CVR.5** à instância (G, k). Se a redução **CVR.5** concluir que a instância é Não, reportamos uma falha. Caso contrário, seja (G', k') a instância reduzida, sobre a qual sabemos que G' tem grau mínimo pelo menos 3 e que $0 \leq k' \leq k$.

Seja R_0 o conjunto de vértices removidos pela regra de redução **CVR.1**, que exclui vértices que possuem laços. Os vértices de R_0 são vértices obrigatórios para um conjunto de vértices de retroalimentação em G (pois laços são ciclos), o que significa que existe um conjunto de vértices de retroalimentação em G de tamanho mínimo possível que contém R_0, e para qualquer conjunto de vértices de retroalimentação R' de G', $R' \cup R_0$ é um conjunto de vértices de retroalimentação de G.

Se G' for um grafo vazio, então devolvemos R_0. Neste caso, observe que $|R_0| \leq k$, uma vez que $k' \geq 0$ e $|R_0| = k - k'$, e R_0 é um conjunto de vértices de retroalimentação de G. Caso contrário, escolhemos uma aresta e de G' uniformemente ao acaso, com probabilidade $1/|E(G')|$, e escolhemos um extremo de e de forma independente ao acaso, que chamaremos de v. Agora, fazemos uma chamada recursiva em $(G' - v, k' - 1)$, devolvendo uma falha se

a etapa recursiva devolver uma falha. Se a etapa recursiva devolver um conjunto de vértices de retroalimentação R', devolveremos $R = R' \cup R_0 \cup \{v\}$.

No segundo caso, o conjunto R' é um conjunto de vértices de retroalimentação de $G' - v$ de tamanho no máximo $k' - 1$. Para provar a corretude do procedimento, note que o limitante de tamanho em R' implica em $|R| \leq k$ e que concluímos que $R' \cup \{v\}$ é um conjunto de vértices de retroalimentação de G', o que faz com que R seja um conjunto de vértices de retroalimentação de G. Assim, o algoritmo sempre reporta uma falha ou devolve um conjunto de vértices de retroalimentação de G de tamanho no máximo k.

O limitante de probabilidade pode ser provado por indução em k. Suponha que exista um conjunto de vértices de realimentação R de G de tamanho no máximo k. Pela análise feita na Seção B.2 sobre as regras de redução, a regra **CVR.5** não será acionada (pois a instância é Sim), a instância (G', k') será computada e existe um conjunto de vértices de retroalimentação R' de G' de tamanho no máximo k'. Se G' é vazio, o algoritmo devolverá R_0. Caso contrário, o grafo G' tem grau mínimo pelo menos 3, e o Lema 38 nos diz que, com probabilidade maior que $1/2$, a aresta e escolhida pelo algoritmo tem pelo menos um extremo em R'. Assim, com probabilidade maior que $1/4$, o vértice v escolhido pelo algoritmo pertence a R', e $R' \setminus \{v\}$ é um conjunto de vértices de retroalimentação de tamanho no máximo $k' - 1$ em $G' - v$. Pela hipótese de indução, a chamada recursiva devolve um conjunto de vértices de retroalimentação $G' - v$ de tamanho no máximo $k' - 1$ com probabilidade pelo menos $4^{-(k'-1)} = 4^{-k'+1}$. É importante ressaltar que o conjunto devolvido não necessariamente será $R' \setminus \{v\}$. Portanto, um conjunto de vértices de retroalimentação de G de tamanho no máximo k é encontrado com probabilidade pelo menos $\frac{1}{4} \cdot 4^{-k'+1} = 4^{-k'} \geq 4^{-k}$, o que conclui a demonstração do teorema. □

Se repetirmos a execução do algoritmo do Teorema 39 independentemente $1/p = 4^k$ vezes, a probabilidade de devolver a resposta correta para instâncias Sim passa a ser constante. Isso nos leva ao seguinte resultado.

Corolário 40 ([3]). *Existe um algoritmo aleatorizado que, dada uma instância (G, k) de* Conjunto de Vértices de Retroalimentação*, ou reporta uma falha ou encontra um conjunto de vértices de retroalimentação em G de tamanho no máximo k em tempo $4^k n^{O(1)}$. Além disso, se o algoritmo receber uma instância* Sim*, ele devolverá uma solução com probabilidade constante.*

C.2 Codificação por cores em Caminho Mais Longo

Aqui, trazemos um exemplo de aplicação da técnica de codificação por cores, abordada na Seção 5.1, em um caso simples, no qual o grafo H é um caminho de k vértices (que chamaremos

de *k-caminho*).

Caminho Mais Longo	
Instância:	Um grafo G.
Parâmetro:	Um inteiro não negativo k.
Questão:	Existe um caminho de k vértices no grafo G?

A ideia é colorir os vértices do grafo aleatoriamente, de maneira uniforme e independente, com um conjunto de cores $[k] = \{1, \ldots, k\}$, e encontrar um k-caminho em que não haja vértices de cores repetidas, caso tal caminho exista.

O lema a seguir é essencial para construir um algoritmo a partir da ideia citada, pois ele nos diz que, se colorirmos os elementos de um conjunto com k cores aleatória e uniformemente, então um subconjunto de k elementos é colorido sem cores repetidas com certa probabilidade.

Lema 41 ([1])**.** *Seja U um conjunto de tamanho n e seja $X \subseteq U$ um subconjunto de tamanho k. Além disso, seja $\chi : U \to [k]$ uma coloração dos elementos de U, tal que cada elemento de U seja colorido aleatoriamente com uma das k cores, de forma independente e uniforme. Então, a probabilidade de que os elementos de X sejam coloridos sem cores repetidas é pelo menos e^{-k}.*

Demonstração. O número total de colorações χ possíveis para U é k^n, e o número de colorações χ em que os elementos de X não têm cores repetidas é $k!k^{n-k}$. Portanto, a probabilidade de que os elementos de X sejam coloridos sem cores repetidas é

$$\frac{k!k^{n-k}}{k^n} = \frac{k!k^n}{k^n k^k} = \frac{k!}{k^k}.$$

Conhecendo a desigualdade $x! > (x/e)^x$, temos que a probabilidade é

$$\frac{k!}{k^k} > \frac{k^k}{k^k e^k} = e^{-k}.$$

□

A etapa de codificação por cores transforma o problema de encontrar um k-caminho em um grafo no problema encontrar um *k-caminho arco-íris* (como chamaremos um caminho cujos vértices não repetem cores) em um grafo colorido nos vértices. Gastaremos, então, um tempo proporcional a 2^k para mantermos a informação sobre quais são cores utilizadas no caminho.

O lema a seguir, e sua demonstração, mostram como podemos obter um algoritmo que encontre, um k-caminho arco-íris no grafo colorido nos vértices, caso exista.

Lema 42 ([1])**.** *Seja G um grafo (orientado ou não orientado), e seja $\chi : V(G) \to [k]$ uma coloração de seus vértices com k cores. Existe um algoritmo que verifica em tempo $2^k n^{O(1)}$ se G contém algum caminho arco-íris de k vértices e, caso contenha, devolve um desses caminhos.*

Demonstração. Seja $V_1, \ldots, V_k$ uma partição de $V(G)$ tal que V_i é o conjunto de todos os vértices de cor i. O algoritmo utilizará a técnica de programação dinâmica. Para um subconjunto não vazio $S \subseteq [k]$ e um vértice u colorido com uma das cores de S, i.e., $\chi(u) \in S$, definimos o valor lógico $\texttt{CAMINHO}(S, u)$, que será verdadeiro se e somente se houver um caminho arco-íris com um extremo em u e contendo todas as cores de S. Quando $|S| = 1$, $\texttt{CAMINHO}(S, u)$ é verdadeiro para qualquer $u \in V(G)$ se e somente se $S = \{\chi(u)\}$. Quando $|S| > 1$, temos a recorrência

$$\texttt{CAMINHO}(S, u) = \begin{cases} \bigvee\{\texttt{CAMINHO}(S \setminus \{\chi(u)\}, v) : uv \in E(G)\} & \text{, se } \chi(u) \in S \\ \text{falso} & \text{, caso contrário,} \end{cases}$$

isto é, se existe um caminho arco-íris com extremo em u contendo todas as cores de S, então existe um caminho arco-íris com extremo em um vizinho v de u contendo todas as cores de $S \setminus \{\chi(u)\}$.

Utilizando essa recorrência, podemos calcular todos os valores possíveis de `CAMINHO` em tempo $2^k n^{O(1)}$. Isso pode ser feito guardando, por exemplo, os valores de `CAMINHO` em uma tabela de dimensão $2^k \times n$, onde cada uma das 2^k linhas representa uma configuração possível de cores utilizadas em S e cada uma das n colunas representa um vértice do grafo. Preenchendo tal tabela a partir das configurações com apenas uma cor em S, é fácil utilizar a recorrência para preencher as linhas correspondentes às configurações com mais cores em S.

Assim, existirá um k-caminho arco-íris em G se e somente se $\texttt{CAMINHO}([k], v)$ for verdadeiro para algum vértice $v \in V(G)$. Caso exista tal caminho, podemos utilizar a tabela citada anteriormente para recuperar os vértices que o formam, começando pela célula $([k], v)$ da tabela e encontrando os vizinhos de v para os quais `CAMINHO` é verdadeiro utilizando $k - 1$ cores. Para cada vizinho, repetimos o procedimento e, como sabemos que existe um k-caminho arco-íris em G, uma das chamadas recursivas desse procedimento encontrará todos os vértices que formam tal caminho. □

Os Lemas 41 e 42 nos permitem obter o seguinte resultado.

Teorema 43 ([1]). *Existe um algoritmo aleatorizado que, dada uma instância (G, k) de* Caminho Mais Longo*, reporta uma falha ou encontra um k-caminho em G em tempo $(2e)^k n^{O(1)}$. Além disso, se o algoritmo receber uma instância* Sim*, ele devolverá uma solução com probabilidade constante.*

Demonstração. O algoritmo descrito no Lema 42 executa em tempo $2^k n^{O(1)}$ e devolve uma solução com probabilidade pelo menos e^{-k}, caso a instância seja Sim. Podemos repetir a execução desse algoritmo e^k vezes, de forma independente, obtendo tempo de execução $e^k 2^k n^{O(1)}$ e probabilidade constante de devolver uma resposta correta para instâncias Sim.

Dada a instância (G, k), a etapa de codificação por cores colore aleatoriamente os vértices de $V(G)$ com as cores $[k]$, de forma independente e uniforme, obtendo a coloração $\chi : V(G) \to [k]$. O algoritmo do Lema 42 é executado, então, no grafo G com a coloração χ. Se ele devolver um k-caminho arco-íris, devolveremos esse caminho como um k-caminho em G. Caso contrário, reportamos uma falha.

Para limitar a probabilidade de encontrar um k-caminho no caso de (G, k) ser uma instância Sim, vamos chamar de P um k-caminho em G, que sabemos que existe, uma vez que a instância é Sim. O Lema 41 nos diz que o caminho P será um k-caminho arco-íris na coloração χ com probabilidade pelo menos e^{-k}. Se P é um k-caminho arco-íris, o algoritmo do Lema 42 encontra um k-caminho arco-íris (que não necessariamente é P, uma vez que uma instância pode ter mais de uma solução), e o devolve como k-caminho em G, o que conclui a prova. □

C.3 Separação aleatória em Subgrafo Isomorfo

A técnica de separação aleatória, abordada na Seção 5.1, é exemplificada no problema Subgrafo Isomorfo.

Subgrafo Isomorfo	
Instância:	Um grafo G de n vértices e um grafo H de k vértices.
Parâmetro:	Um inteiro não negativo k.
Questão:	Existe um subgrafo de G isomorfo a H?

Note que o problema Caminho Mais Longo, tratado na Seção C.2, é um caso especial do problema Subgrafo Isomorfo, onde o grafo H é um k-caminho.

Mostraremos, a seguir, que Subgrafo Isomorfo pode ser resolvido em tempo $f(d, k)n^{O(1)}$ para alguma função computável f, quando o grau de G é limitado por d. Isto significa que Subgrafo Isomorfo é FPT quando parametrizado por k e d [8].

A ideia essencial por trás da técnica é colorir aleatoriamente as arestas (resp., os vértices) do grafo, de modo que as arestas (resp., os vértices) do subgrafo procurado tenham uma cor e as arestas (resp., os vértices) adjacentes às arestas (resp., aos vértices) desse subgrafo tenham outra cor. Daí vem o nome *separação*: separamos o subgrafo procurado de suas arestas/vértices vizinhos usando uma coloração aleatória. No problema Subgrafo Isomorfo, a ideia geral é que se conseguirmos realizar certa coloração, cada componente conexo do grafo H corresponderá a um componente conexo do subgrafo induzido pela primeira cor no grafo colorido G.

A primeira coisa a se fazer é verificar se o grau máximo de H é no máximo d, pois, em caso negativo, sabemos que estamos lidando com uma instância Não. Em caso positivo, colorimos aleatoriamente, de forma independente, cada uma das arestas de G com uma de duas cores, que podem ser vermelho ($\mathtt{V}$) ou azul ($\mathtt{A}$), cada uma com probabilidade 1/2. A coloração aleatória obtida será $\chi\colon E(G) \to \{\mathtt{V}, \mathtt{A}\}$.

Suponhamos que (G, H) é uma instância Sim, i.e., existe um subgrafo $\widehat{H}$ de G isomorfo a H. Denotaremos por J o conjunto de arestas incidentes a algum vértice de $V(\widehat{H})$, mas que não pertencem a $E(\widehat{H})$. Chamaremos uma coloração χ de *bem-sucedida* se ambas as condições a seguir forem válidas: (i) toda aresta de $E(\widehat{H})$ é vermelha, e (ii) toda aresta de J é azul.

Sabendo que toda aresta de $E(\widehat{H}) \cup J$ é incidente a um vértice de $V(\widehat{H})$, que $|V(\widehat{H})| = |V(H)| = k$, e que o grau máximo de G é limitado superiormente por d, obtemos a seguinte relação:

$$|E(\widehat{H})| + |J| \leq d|V(\widehat{H})| \leq dk.$$

Com isso, a probabilidade de que a coloração χ seja bem-sucedida é pelo menos

$$\frac{1}{2^{|E(\widehat{H})|+|J|}} \geq \frac{1}{2^{dk}}.$$

Agora, seja $G_{\mathtt{V}}$ o subgrafo de G contendo apenas as arestas vermelhas. Mostraremos que se a coloração χ for bem-sucedida, então $\widehat{H}$ é um subgrafo de $G_{\mathtt{V}}$ e, além disso, $\widehat{H}$ consiste exatamente em um certo número de componentes conexos de $G_{\mathtt{V}}$.

Consideremos, primeiro, o caso em que H é conexo. Dessa forma, a coloração χ ser bem-sucedida implica que $\widehat{H}$ é um dos componentes conexos de $G_{\mathtt{V}}$. Basta, então, percorrermos todos os componentes conexos de $G_{\mathtt{V}}$ que tenham exatamente k vértices, verificando quais deles são isomorfos a H. Essa verificação pode ser feita por força bruta em tempo $k!k^{O(1)}$: podemos testar todas as $k!$ permutações dos vértices de um componente conexo, associando

cada vértice da permutação a um vértice de H e, para cada permutação, verificar se as arestas entre seus vértices são equivalentes às arestas entre os vértices de H – o que, para cada permutação, leva tempo polinomial em k.

Observação 2. Existe um algoritmo mais complexo para verificar se dois grafos são isomorfos, com tempo de execução $k^{O(d \log d)}$ em grafos de k vértices e grau máximo d. Tal algoritmo foi proposto por Luks, em 1982 [22]. Mesmo que o funcionamento de tal algoritmo não seja de nosso interesse, podemos considerar sua utilização no lugar do algoritmo de força bruta, para fins teóricos.

Partimos, então, para o caso em que H não é conexo. Denotamos por $H_1, H_2, \ldots, H_p$ os componentes conexos de H e por $C_1, C_2, \ldots, C_r$ os componentes conexos de $G_{\mathtt{v}}$. Construímos um grafo bipartido B, no qual todo vértice de $V(B)$ corresponde a um componente conexo H_i ou a um componente conexo C_j. Nesse grafo, H_i será adjacente a C_j se ambos os componentes conexos correspondentes a esses vértices têm no máximo k vértices e são isomorfos. Note que a construção do grafo B leva tempo $k!k^{O(1)}n^{O(1)}$ ou tempo $k^{O(d \log d)}n^{O(1)}$, dependendo do algoritmo utilizado para verificar o isomorfismo. Com essa construção, o subgrafo $\widehat{H}$ isomorfo a H corresponde a um emparelhamento no grafo B que satura $\{H_1, H_2, \ldots, H_p\}$. Todo componente de B é um grafo bipartido, e é sabido que existe um algoritmo polinomial para encontrar um emparelhamento nessa classe de grafos. Portanto, é possível verificar se existe um subgrafo de $G_{\mathtt{v}}$ isomorfo a H em tempo $k!k^{O(1)}n^{O(1)}$ ou em tempo $k^{O(d \log d)}n^{O(1)}$, dependendo do algoritmo utilizado para a verificação do isomorfismo.

Como sabemos que a probabilidade de que χ seja bem-sucedida é pelo menos 2^{-dk}, podemos repetir 2^{dk} vezes a execução do algoritmo descrito acima, obtendo uma probabilidade constante de devolver a resposta correta para instâncias Sim. Isso é descrito no resultado a seguir.

Teorema 44 ([8])**.** *Dada uma instância (G, H) de* Subgrafo Isomorfo*, onde $|V(G)| = n$, $|V(H)| = k$, e o grau de cada vértice de G é no máximo d, existem algoritmos que a resolvem em tempo $2^{dk}k!k^{O(1)}n^{O(1)}$ e em tempo $2^{dk}k^{O(d \log d)}n^{O(1)}$. Além disso, se a instância (G, H) é* Sim*, tais algoritmos devolvem uma solução com probabilidade constante.*

D Exercícios resolvidos

Uma parte do trabalho realizado no período de pesquisa consistiu na resolução de exercícios propostos ao final dos capítulos estudados do livro de referência [11].

Nesta seção, enunciamos os problemas estudados como exercícios, bem como a tarefa proposta para cada um deles, com as respectivas resoluções.

D.1 *Kernelização* em Cobertura de Pontos por Linhas

Considere o seguinte problema.

Cobertura de Pontos por Linhas	
Instância:	Um conjunto P de pontos no plano.
Parâmetro:	Um inteiro k.
Questão:	Existe um conjunto L de no máximo k linhas no plano tal que todo ponto em P esteja sobre alguma linha de L?

Para Cobertura de Pontos por Linhas, o exercício foi provar o seguinte resultado sobre *kernelização*.

Teorema 45. Cobertura de Pontos por Linhas *admite um kernel com* $O(k^2)$ *pontos.*

Demonstração. Dada uma instância (P, k) de Cobertura de Pontos por Linhas, considere as seguintes regras de redução.

Redução CPL.1 Se existe uma linha ℓ que contém mais que k pontos de P, remova de P os pontos cobertos por ℓ e decremente o parâmetro k em 1. A nova instância será $(P \setminus S_\ell, k-1)$, onde S_ℓ denota o conjunto dos pontos de P cobertos por ℓ.

Segurança da regra. Se (P, k) é uma instância Sim, então existe um conjunto L de linhas no plano tal que $|L| \leq k$ e todo ponto de P é coberto por alguma linha de L. Como ℓ cobre mais que k pontos de P, então certamente $\ell \in L$, pois, caso contrário, precisaríamos incluir em L pelo menos $k+1$ linhas para cobrir os mesmos pontos que ℓ cobre, o que não é possível, uma vez que (P, k) é uma instância Sim. Assim, $(P \setminus S_\ell, k-1)$ é uma instância Sim, uma vez que $L' = L \setminus \{\ell\}$ é um conjunto de linhas tal que $|L'| \leq k-1$ e todo ponto em $P \setminus S_\ell$ é coberto por alguma linha de L'.

Na direção oposta, se $(P \setminus S_\ell, k-1)$ é uma instância Sim, então existe um conjunto L' de linhas tal que $|L'| \leq k-1$ e todo ponto de $P \setminus S_\ell$ é coberto por alguma linha de L'. Note

que $L = L' \cup \{\ell\}$ é um conjunto de linhas tal que $|L| \leq k$ e todo ponto de $P = (P \setminus S_\ell) \cup S_\ell$ é coberto por alguma linha de L. Portanto, (P, k) é uma instância SIM. □

Redução CPL.2 Seja (P, k) uma instância na qual a regra **CPL.1** não é aplicável. Se $k < 0$ ou P tem mais que k^2 pontos, então concluímos que (P, k) é uma instância NÃO.

Segurança da regra. Uma vez que a regra **CPL.1** não é aplicável a (P, k), sabemos que qualquer linha no plano contém no máximo k pontos de P. Assim, se $|P| > k^2$, é impossível cobrir todos os pontos de P utilizando k linhas. Portanto, se $|P| \leq k^2$ concluímos que (P, k) é uma instância NÃO.

Note que $k < 0$ pode ocorrer ao aplicarmos a regra **CPL.1** a uma instância $(P, 0)$, o que significa que a capacidade do parâmetro já foi ultrapassada. Portanto, se $k < 0$, também concluímos que (P, k) é uma instância NÃO. □

Ambas as regras de redução podem ser aplicadas em tempo polinomial em $|P| = n$. A redução **CPL.1** pode ser aplicada escolhendo cada um dos $\binom{n}{2} = O(n^2)$ pares de pontos e, para cada par, traçar a reta que passa por ambos. Para cada reta traçada, basta verificar quantos pontos de P a reta cobre. Da forma mais ingênua, isso pode ser feito em tempo $O(n^3)$. Já a redução **CPL.2** pode ser aplicada em tempo constante, se considerarmos que $|P| = n$ nos é dado junto ao conjunto P na entrada.

Após a aplicação exaustiva da redução **CPL.1**, ou poderemos aplicar a regra **CPL.2**, concluindo que estamos lidando com uma instância NÃO, ou teremos um *kernel* com no máximo k^2 pontos. □

D.2 Caracterização de um grafo *cluster*

Dizemos que um grafo G é um *cluster* se toda componente conexa de G é uma clique.

A primeira tarefa com relação a este conceito foi provar o teorema abaixo, que nos dá uma caracterização para um dado grafo ser um *cluster*.

Teorema 46. *Um grafo G é um cluster se e somente se ele não contém um caminho induzido em três vértices, i.e., se não existem três vértices $u, v, w \in V(G)$ tais que $uv, vw \in E(G)$ e $wu \notin E(G)$.*

Demonstração. ($\implies$) Seja G um grafo *cluster* de n vértices. Se $n < 3$, claramente não é possível que G contenha um caminho induzido em três vértices. Sejam, então, $n \geq 3$ e $u, v, w \in V(G)$.

Consideraremos três casos:

- Se u, v e w estão em três componentes distintas, então o subgrafo induzido por u, v e w consiste em três vértices isolados.
- Se dois entre u, v e w estão na mesma componente e o terceiro está em uma componente distinta dos demais, então o subgrafo induzido por u, v e w consiste em dois vértices ligados por uma aresta e mais um vértice isolado.
- Se u, v e w estão todos na mesma componente, então o subgrafo induzido por u, v e w consiste em um grafo completo de três vértices (um triângulo), uma vez que tal componente é uma clique.

Note que, em todos os casos, o subgrafo induzido por três vértices quaisquer de G não é um caminho entre os três vértices e, portanto, G não contém um caminho induzido em três vértices.

($\Longleftarrow$) Seja G um grafo que não contém um caminho induzido em três vértices. Se G não possui arestas, claramente G é um *cluster*, uma vez que toda componente conexa de G seria um subgrafo completo de apenas um vértice.

Consideremos, então, que G tem pelo menos uma aresta e que u e v são dois vértices adjacentes em G. Analisaremos dois casos:

- Quando ambos u e v possuem grau 1, i.e., cada um deles só tem o outro como vizinho. Neste caso, a componente de u e v consiste apenas na aresta entre eles.
- Quando pelo menos um entre u e v possui um vizinho. Neste caso, suponha, sem perda de generalidade, que o vértice u possui um vizinho diferente de v, que chamaremos de w. Como G não contém um caminho induzido em três vértices, sabemos que o subgrafo de G induzido por u, v e w contém as arestas uv e uw e também deve conter a aresta vw, pois, caso não contivesse, o grafo G possuiria um caminho induzido nesses três vértices. Assim, se u e v são adjacentes e w é adjacente a u, então w também terá de ser adjacente a v e, portanto, se escolhermos quaisquer três vértices de uma componente, o subgrafo induzido por eles será um triângulo.

Em ambos os casos, cada vértice é adjacente a todos os outros vértices da mesma componente e, portanto, toda componente de G é uma clique. Por definição, G é um *cluster*. □

D.3 *Kernelização* em Edição em Cluster

Denotamos por $X \triangle Y$ a *diferença simétrica* entre dois conjuntos X e Y, definida como $X \triangle Y = (X \setminus Y) \cup (Y \setminus X)$.

Considere o seguinte problema.

Edição em Cluster	
Instância:	Um grafo G.
Parâmetro:	Um inteiro k.
Questão:	Existe um conjunto $A \subseteq \binom{V(G)}{2}$ de tamanho no máximo k tal que o grafo $(V(G), E(G) \triangle A)$ é um *cluster*?

Em outras palavras, Edição em Cluster consiste em decidir se é possível transformar o grafo G de entrada em um *cluster* realizando no máximo k operações de inserção ou remoção de arestas em G.

Para Edição em Cluster, tivemos que demonstrar o seguinte resultado de *kernelização*.

Teorema 47. Edição em Cluster *admite um kernel com* $O(k^2)$ *vértices.*

Demonstração. Dada uma instância (G, k), considere as seguintes regras de redução:

Redução EC.1 Se existe um vértice $u \in V(G)$ que não faz parte de nenhum caminho induzido em três vértices, então remova u de G. A nova instância será $(G - u, k)$.

Segurança de regra. Se (G, k) é uma instância Sim, então existe um conjunto $A \subseteq \binom{V(G)}{2}$ tal que $|A| \leq k$ e o grafo $(V(G), E(G) \triangle A)$ é um *cluster*. Como o objetivo do problema é remover todos os caminhos induzidos em três vértices do grafo de entrada, se o subgrafo induzido por três vértices arbitrários não forma um caminho, não é necessário incluir ou remover arestas entre tais vértices, pois não há caminho induzido entre eles para removermos do grafo. Assim, o fato de u não participar de nenhum caminho induzido em três vértices nos faz concluir que nenhuma aresta do conjunto A tem u como um de seus extremos. Dessa forma, o conjunto A faz com que o grafo $(V(G - u), E(G - u) \triangle A)$ também não possua caminhos induzidos em três vértices. Como $|A| \leq k$, a instância $(G - u, k)$ é uma instância Sim.

Na direção oposta, se $(G - u, k)$ é uma instância Sim, então existe um conjunto $A \subseteq \binom{V(G-u)}{2}$ tal que $|A| \leq k$ e o grafo $(V(G - u), E(G - u) \triangle A)$ é um *cluster*.

Podemos observar que o conjunto A também fará com que $(V(G), E(G) \triangle A)$ seja um *cluster*, pois se o grafo $(V(G - u), E(G - u) \triangle A)$ não contém caminhos induzidos em três vértices, o fato de A não conter arestas adjacentes ao vértice u faz com que $(V(G), E(G) \triangle A)$ também não contenha tais caminhos, pois u não participa de nenhum caminho induzido em três vértices. Como $|A| \leq k$, a instância (G, k) é uma instância Sim. □

Redução EC.2 Se existem dois vértices adjacentes u e v tais que a aresta uv faz parte de pelo menos $k + 1$ caminhos induzidos em três vértices diferentes, então remova uv de G e decremente k em 1. A nova instância será $(G - uv, k - 1)$.

Segurança de regra. Se (G, k) é uma instância Sim, então existe um conjunto $A \subseteq \binom{V(G)}{2}$ tal que $|A| \leq k$ e o grafo $(V(G), E(G) \triangle A)$ é um *cluster*. A aresta uv certamente está contida no conjunto A, pois, caso contrário, seria necessário incluir em A uma aresta de cada um dos pelo menos $k+1$ caminhos induzidos em três vértices dos quais uv faz parte. Porém, como sabemos que (G, k) é uma instância Sim, não é possível que A contenha mais que k arestas e, portanto, podemos afirmar que $uv \in A$.

Dessa forma, como $(V(G), E(G) \triangle A)$ é um *cluster*, o conjunto $A' = A \setminus \{uv\}$ fará com que o grafo $(V(G - uv), E(G - uv) \triangle A')$ também seja um cluster. Como $|A'| \leq k - 1$, a instância $(G - uv, k - 1)$ é uma instância Sim.

Na direção oposta, se $(G - u, k - 1)$ é uma instância Sim, então existe um conjunto $A' \subseteq \binom{V(G-u)}{2}$ tal que $|A'| \leq k - 1$ e o grafo $(V(G - u), E(G - u) \triangle A')$ é um *cluster*. Note que o conjunto $A = A' \cup \{uv\}$ faz com que $(V(G), E(G) \triangle A)$ também não contenha caminhos induzidos em três vértices, pois a aresta uv desfaz quaisquer tais caminhos que possam existir em G e não em $G - uv$. Uma vez que $|A| \leq k$, concluímos que (G, k) é uma instância Sim. □

Redução EC.3 Se existem dois vértices não adjacentes u e v tais que pelo menos $k + 1$ caminhos induzidos em três vértices diferentes contêm ambos u e v, então adicione a aresta uv em G e decremente k em 1. A nova instância será $(G + uv, k - 1)$.

Segurança de regra. A demonstração de segurança desta regra é análoga à demonstração para a regra **EC.2**. Aqui, para um grafo G, se $u, v \in V(G)$ e $uv \notin E(G)$, chamaremos uv de *não-aresta*.

Se (G, k) é uma instância Sim, então existe um conjunto $A \subseteq \binom{V(G)}{2}$ tal que $|A| \leq k$ e o grafo $(V(G), E(G) \triangle A)$ é um *cluster*. A não-aresta uv certamente está contida no conjunto A, pois, caso contrário, seria necessário incluir em A uma aresta de cada um dos pelo menos $k+1$ caminhos induzidos em três vértices dos quais uv faz parte. Porém, como sabemos que (G, k) é uma instância Sim, não é possível que A contenha mais que k arestas e, portanto, podemos afirmar que $uv \in A$. Dessa forma, como $(V(G), E(G) \triangle A)$ é um *cluster*, o conjunto $A' = A \setminus \{uv\}$ fará com que o grafo $(V(G + uv), E(G + uv) \triangle A')$ também seja um cluster. Como $|A'| \leq k - 1$, a instância $(G + uv, k - 1)$ é uma instância Sim.

Na direção oposta, seja $G' = G + uv$. Se $(G', k-1)$ é uma instância Sim, então existe um conjunto $A' \subseteq \binom{V(G')}{2}$ tal que $|A'| \leq k - 1$ e o grafo $(V(G'), E(G') \triangle A')$ é um *cluster*.

Note que se A' faz com que $(V(G'), E(G') \triangle A')$ não contenha caminhos induzidos em três vértices, então o conjunto $A = A' \cup \{uv\}$ faz com que $(V(G), E(G) \triangle A)$ também não contenha tais caminhos, pois a remoção da aresta uv em G' pode ter criado caminhos induzidos em três vértices em G, mas todos esses caminhos serão removidos de $(V(G), E(G) \triangle A)$ se incluirmos uv no conjunto A. Uma vez que $|A| \leq k$, concluímos que (G, k) é uma instância Sim. □

Com um argumento mais complexo e extenso do que os utilizados nos outros resultados estudados de kernelização através de regras de redução, que omitimos aqui, é possível demonstrar que se nenhuma das regras **EC.1**, **EC.2** ou **EC.3** são aplicáveis a uma instância, então podemos ou identificá-la como uma instância NÃO ou obter um *kernel* com $O(k^3)$ vértices e $O(k^2)$ arestas.

□

Não incluímos a demonstração do último argumento no relatório por falta de tempo, mas mantivemos as demonstrações de segurança das regras por julgarmos interessante abordar o mesmo problema através de duas técnicas distintas, como veremos na subseção seguinte.

D.4 Árvores de busca em Edição em Cluster

Ainda sobre Edição em Cluster, tivemos a tarefa de utilizar a técnica de árvores de busca limitada para encontrar um algoritmo FPT para o problema.

Teorema 48. *Existe um algoritmo de tempo* $3^k n^{O(1)}$ *para* Edição em Cluster*, baseado na técnica de árvores de busca limitadas.*

Demonstração. Dada uma instância (G, k), como já comentamos, o objetivo do problema é decidir se é possível remover todos os caminhos induzidos em três vértices de G realizando no máximo k operações de inserção ou remoção de arestas em G.

Note que, se tomarmos três vértices $u, v, w \in V(G)$, o subgrafo induzido por eles só será um caminho se existirem dois pares adjacentes entre eles e um par não adjacente, por exemplo, e sem perda de generalidade, se $uv, vw \in E(G)$ e $wu \notin E(G)$.

Se o subgrafo induzido por três vértices u, v e w é um caminho, há três formas de fazer com que tal subgrafo deixe de ser um caminho a partir de operações de inserção ou remoção de arestas, considerando que $uv, vw \in E(G)$ e $wu \notin E(G)$:

- Inserir a aresta uw, de modo que o subgrafo induzido passe a ser um triângulo.
- Excluir a aresta uv, de modo que o subgrafo induzido passe a ter um vértice isolado u e uma aresta isolada vw.
- Excluir a aresta vw, de modo que o subgrafo induzido passe a ter um vértice isolado u e uma aresta isolada vw.

Podemos, então, projetar um algoritmo de ramificação que identifique um caminho induzido em três vértices e ramifique em cada um dos três casos citados. Uma vez que o parâmetro k é justamente o número máximo de operações de inserção ou remoção de arestas que podemos realizar, a altura da árvore de busca de tal algoritmo será no máximo k. Portanto, o algoritmo resolverá 3^k subproblemas, cada um deles sendo um nó de sua árvore de busca.

Para $|V(G)| = n$ que existem $\binom{n}{3} = O(n^3)$ subgrafos induzidos em três vértices e, para cada um deles, podemos verificar se é um caminho induzido em tempo constante. Além disso, para cada caminho induzido em três vértices, efetuar uma operação de inserção ou remoção de aresta também pode ser feito em tempo constante. Se, em algum dos 3^k nós da árvore de busca, identificarmos que não existem caminhos induzidos em três vértices no grafo atual, devolvemos que a instância é SIM. Se em nenhum deles conseguirmos zerar o número de caminhos induzidos em três vértices, devolvemos que a instância é NÃO.

Portanto, podemos construir tal algoritmo baseado em árvores de busca limitadas em tempo $3^k \cdot O(n^3)$. □

D.5 *Kernelização* em Subgrafo Ímpar com k Arestas

Considere o seguinte problema.

Subgrafo Ímpar com k Arestas	
Instância:	Um grafo G.
Parâmetro:	Um inteiro k.
Questão:	O grafo G contém um subgrafo H com exatamente k arestas tal que todos os vértices de H tenham grau ímpar?

Para Subgrafo Ímpar com k Arestas, também tivemos de demonstrar um resultado de *kernelização*.

Teorema 49. Subgrafo Ímpar com k Arestas *admite um kernel com $O(k^2)$ vértices.*

Demonstração. Seja (G, k) uma instância do problema. Como vértices isolados não participam de nenhum subgrafo ímpar de G, podemos considerar em nossa análise que G não possui vértices isolados. Podemos analisar os três casos a seguir:

1. Se o grafo G tem grau máximo k e possui pelo menos $2k^2$ arestas, então G certamente contém um emparelhamento de tamanho k, que claramente é um subgrafo ímpar com k arestas. Isso é possível pelo fato de que, se todo vértice de G tem no máximo k vizinhos e G tem pelo menos $k(2k) = 2k^2$ arestas, então, pelo Princípio da Casa dos Pombos, existem k pares disjuntos de vértices adjacentes. Assim, neste caso, se $E(G) \geq 2k^2$, podemos afirmar que (G, k) é uma instância Sim. Se $E(G) < 2k^2$, então temos um *kernel* com no máximo $2k^2 - 1 = O(k^2)$ arestas e, consequentemente, $O(k^2)$ vértices.
2. Se G possui pelo menos um vértice v com grau maior que k e o parâmetro k é ímpar, então uma estrela com v como vértice interno e com k folhas será um subgrafo ímpar com k arestas e, portanto, (G, k) será uma instância Sim.

3. Se G possui pelo menos um vértice v com grau maior que k e o parâmetro k é par, consideramos dois subcasos:
   (a) Se G é uma estrela com v como vértice interno, então (G, k) é uma instância NÃO.
   (b) Se G não é uma estrela, então existe pelo menos uma aresta $uw \in E(G)$, para dois vértices u e w diferentes de v. Uma vez que v tem mais que k vizinhos, existem pelo menos $k-1$ arestas vz, para vértices z diferentes de u e y. O subgrafo induzido por essas $k-1$ arestas vz e pela aresta uw, com k sendo par, é subgrafo ímpar com k arestas. Portanto, neste caso, (G, k) é uma instância SIM.

É fácil observar que todos os casos podem ser testados em tempo polinomial.

O único caso em que não conseguimos afirmar se a instância é SIM ou NÃO é quando G tem grau máximo k mas possui menos que $2k^2$ arestas. Porém, neste caso, obtemos um *kernel* com $O(k^2)$ vértices. □

D.6 Algoritmo FPT para Clique e Conjunto Independente

Considere o seguintes problema.

Clique	
Instância:	Um grafo G.
Parâmetro:	Um inteiro k.
Questão:	Existe uma clique de k vértices em G?

Para CLIQUE, a tarefa foi mostrar que o problema é FPT quando o grafo de entrada é r-regular (grafo com que todos os vértices têm grau r), para todo inteiro fixo r.

Demonstração. Seja (G, k) uma instância de CLIQUE tal que G é r-regular.

A primeira observação a se fazer é que podemos considerar que $k \leq r + 1$, pois, caso contrário, (G, k) seria uma instância NÃO, uma vez que em um grafo r-regular, a maior clique possível teria $r + 1$ vértices, e o parâmetro k seria maior que $r + 1$.

Considerando que $k \leq r + 1$ e $V(G) = n$, podemos verificar, para cada um dos n vértices $v \in V(G)$ se existe conjunto S de $k-1$ vizinhos de v tais que o subgrafo induzido por $S \cup \{v\}$ é uma clique.

O número de conjuntos S possíveis de $k-1$ elementos entre os vizinhos de v é $\binom{r}{k-1}$ e, para cada conjunto S, podemos verificar se $S \cup \{v\}$ é uma clique em tempo $O(k^2)$. Como tal verificação é feita para cada um dos n vértices de G, o algoritmo tem tempo de execução total $O(k^2) \cdot \binom{r}{k-1} \cdot n$ e, portanto, é FPT sobre o parâmetro k. □

Agora, considere o seguinte problema.

Conjunto Independente	
Instância:	Um grafo G.
Parâmetro:	Um inteiro k.
Questão:	Existe um conjunto X de k vértices em G tal que $uv \notin E(G)$, para todo $u, v \in X$?

Para Conjunto Independente, mostramos que o problema é equivalente a Clique e, portanto, também FPT se o grafo de entrada é r-regular.

Demonstração. Seja (G, k) uma instância de Clique. Se (G, k) é uma instância Sim, então existe uma clique Q de k vértices em G. No grafo $\overline{G}$, complementar de G, sabemos que não existirão arestas entre nenhum dos vértices de Q. Portanto, os vértices de Q formam um conjunto independente de k vértices em $\overline{G}$ e, consequentemente, $(\overline{G}, k)$ é uma instância Sim de Conjunto Independente.

Na direção oposta, seja (G, k) uma instância de Conjunto Independente. Se (G, k) é uma instância Sim, então existe um conjunto independente S de k vértices em G. No grafo $\overline{G}$, sabemos que deverão existir arestas entre todos os vértices de S. Portanto, os vértices de S formam uma clique de k vértices em $\overline{G}$ e, consequentemente, $(\overline{G}, k)$ é uma instância Sim de Clique.

Assim, como ambos os problemas são equivalentes, dada uma instância (G, k) de Conjunto Independente, podemos executar o algoritmo FPT para **Clique** na instância $(\overline{G}, k)$. A instância (G, k) será Sim se e somente $(\overline{G}, k)$ for Sim. □

D.7 Árvores de busca em Remoção de Vértices em Cluster

Considere o seguinte problema.

Remoção de Vértices em Cluster	
Instância:	Um grafo G.
Parâmetro:	Um inteiro k.
Questão:	Existe um conjunto X de no máximo k vértices de G tal que $G - X$ é um *cluster*? Aqui, um *cluster* é um grafo no qual toda componente conexa é uma clique.

Como prática da técnica de árvores de busca limitadas, tivemos a tarefa de encontrar um algoritmo de tempo $3^k n^{O(1)}$ para Remoção de Vértices em Cluster.

Demonstração. A observação importante para este resultado é que podemos construir um

algoritmo de funcionamento análogo ao algoritmo para Edição em Cluster.

Dada uma instância (G, k) de Remoção de Vértices em Cluster, o objetivo do problema é decidir se é possível remover todos os caminhos induzidos em três vértices de G realizando no máximo k operações de remoção de vértices em G.

Note que, se tomarmos três vértices $u, v, w \in V(G)$, o subgrafo induzido por eles só será um caminho se existirem dois pares adjacentes entre eles e um par não adjacente, por exemplo, e sem perda de generalidade, se $uv, vw \in E(G)$ e $wu \notin E(G)$.

Se o subgrafo induzido por três vértices u, v e w é um caminho, há três formas de fazer com que tal subgrafo deixe de ser um caminho a partir de operações de remoção de vértices, considerando que $uv, vw \in E(G)$ e $wu \notin E(G)$: remover u ou remover v ou remover w. Em todos os três casos, o caminho induzido por eles será removido de G após a operação de remoção.

Podemos, então, projetar um algoritmo de ramificação que identifique um caminho induzido em três vértices e ramifique em cada um dos três casos citados. Uma vez que o parâmetro k é justamente o número máximo de operações de remoção de vértices que podemos realizar, a altura da árvore de busca de tal algoritmo será no máximo k. Portanto, o algoritmo resolverá 3^k subproblemas, cada um deles sendo um nó de sua árvore de busca.

Para $|V(G)| = n$ que existem $\binom{n}{3} = O(n^3)$ subgrafos induzidos em três vértices e, para cada um deles, podemos verificar se é um caminho induzido em tempo constante. Além disso, para cada caminho induzido em três vértices, efetuar uma operação de remoção de vértice também pode ser feito em tempo constante. Se, em algum dos 3^k nós da árvore de busca, identificarmos que não existem caminhos induzidos em três vértices no grafo atual, devolvemos que a instância é Sim. Se em nenhum deles conseguirmos zerar o número de caminhos induzidos em três vértices, devolvemos que a instância é Não.

Portanto, podemos construir tal algoritmo baseado em árvores de busca limitadas em tempo $3^k \cdot O(n^3)$. □

D.8 Algoritmo aleatorizado para Empacotamento de Triângulos

Considere o seguinte problema.

Empacotamento de Triângulos	
Instância:	Um grafo G.
Parâmetro:	Um inteiro positivo k.
Questão:	Existem k triângulos disjuntos em vértices em G?

Para Empacotamento de Triângulos, o objetivo era mostrar que o problema admite um algoritmo aleatorizado FPT utilizando a técnica de codificação por cores.

Demonstração. Seja (G, k) uma instância de Empacotamento de Triângulos. Suponha que existe um subconjunto $S \subseteq V$ de tamanho $3k$ que induz k triângulos disjuntos em vértices. Seja $\chi\colon V(G) \to [3k]$ uma coloração dos vértices de G com $3k$ cores (que podem se repetir). Pelo Lema 41, χ colore os elementos de S sem repetição de cores com probabilidade pelo menos e^{-3k}.

A ideia, então, é projetar um algoritmo para verificar se o grafo G contém k triângulos disjuntos em vértices tais que os $3k$ vértices de tais triângulos não repitam cores entre si, i.e., sejam coloridos com $3k$ cores diferentes.

Dadas as $3k$ cores de χ (que podem se repetir), vamos contar quantas são as formas diferentes de separá-las em k *bags* de 3 elementos. Podemos permutar as $3k$ cores (as vendo como elementos distintos) de $(3k)!$ formas. Para cada uma dessas permutações, se a dividirmos em k *bags* de 3 elementos faz com que a permutação das k *bags* não altere a solução, nos permitindo dividir o número de permutações por $k!$. Do mesmo modo, dentro de cada uma das k *bags*, a permutação de seus 3 elementos também não altera a solução, nos permitindo dividir o número de permutações por $(3!)^k = 6^k$. Portanto, podemos separar as $3k$ cores em k *bags* de $\frac{(3k)!}{6^k k!}$ formas diferentes.

Para cada *bag* de 3 cores, podemos verificar se G possui um triângulo com exatamente as 3 cores da *bag* em tempo $O(n^3)$ e, como existem k *bags*, a verificação para todas elas pode ser feita em tempo $O(k \cdot n^3)$.

Dessa forma, o algoritmo projetado consegue, em tempo $O(k \cdot n^3) \cdot \frac{(3k)!}{6^k k!}$, verificar se existem k triângulos disjuntos nas cores de χ, ou seja, todos $3k$ vértices dos triângulos possuem cores distintas. Claramente, se isso for verdade, os k triângulos encontrados são disjuntos em vértices e, portanto, a instância (G, k) é Sim. Caso contrário, a instância Não.

É importante destacar que o algoritmo projetado, por ser aleatorizado, devolve em tempo $O(k \cdot n^3) \cdot \frac{(3k)!}{6^k k!}$ uma solução com probabilidade pelo menos e^{-3k}, caso a instância seja Sim. Então, podemos repetir a execução desse algoritmo e^{3k} vezes, de forma independente, obtendo tempo de execução $e^{3k} \cdot O(k \cdot n^3) \cdot \frac{(3k)!}{6^k k!}$ e probabilidade constante de devolver uma resposta correta para instâncias Sim.

Uma vez que $e^{3k} \cdot O(k \cdot n^3) \cdot \frac{(3k)!}{6^k k!} = f(k) \cdot n^3$, para $f(k) = e^{3k} \cdot k \cdot \frac{(3k)!}{6^k k!}$, tal algoritmo é um algoritmo FPT aleatorizado para Empacotamento de Triângulos. □