Chapter 5

Uma introdução à complexidade parametrizada

Vinicius Fernandes dos Santos, Uéverton dos Santos Souza

Abstract

The solution to many real problems often requires an algorithmic approach, with the subsequent implementation in some system. Typically both the data volume and the frequency of accesses to the system are great, and hence it is necessary an efficient *algorithm (traditionally of polynomial time). The theory of NP-completeness was developed to determine which problems probably can not be solved by polynomial algorithms. However, as many NP-hard problems need to be solved in practice, one possibility is to resort to an approximate or heuristic algorithm instead of an exact algorithm. A recent and promising alternative for the treatability of these problems is to use an analysis from the point of view of* Parameterized Complexity Theory. *This theory, developed by Downey and Fellows, studies the existence of algorithms whose exponential complexity depends only on certain aspects of the input, such algorithms are called fixed parameter tractable (or simply FPT algorithms). This course will introduce the formal concepts of parameterized complexity, basic techniques for designing FPT algorithms, such as bounded search trees and kernelization, and also techniques for negative results, for problems where FPT algorithms are not expected to exist. The examples used will be focused in graph problems.*

Resumo

A solução para diversos problemas reais frequentemente exige uma abordagem algorítmica, com a posterior implementação em algum sistema. Tipicamente tanto o volume de dados como a frequência de acessos ao sistema são grandes, sendo portanto necessário algoritmos eficientes *(tradicionalmente de tempo polinomial). A teoria da NP-completude foi desenvolvida para determinar quais problemas provavelmente não podem ser resolvidos por algoritmos polinomiais. Entretanto, como muitos problemas NP-difíceis precisam ser resolvidos na prática, uma possibilidade é recorrer a um algoritmo aproximativo ou heurístico em vez de um algoritmo exato. Uma recente e promissora alternativa para a tratabilidade desses problemas, é recorrer a uma análise sob o ponto*

de vista da Teoria da Complexidade Parametrizada. *Esta teoria, desenvolvida por Downey e Fellows, estuda a existência de algoritmos cuja complexidade exponencial depende apenas de certos aspectos da entrada, tais algoritmos são denominados tratáveis por parâmetro fixo (ou simplesmente algoritmos FPT). Este curso, introduzirá os conceitos formais da complexidade parametrizada, técnicas básicas de desenvolvimento de algoritmos FPT, tais como árvores de altura limitada e redução a um núcleo, e também técnicas para resultados negativos, para problemas onde não se espera ser possível desenvolver algoritmos FPT. Os exemplos utilizados serão focados em problema em grafos.*

5.1. Introdução

A questão "$P = NP$?" é a mais importante questão em aberto da ciência da computação. E a teoria da NP-completude foi desenvolvida para mostrar que determinados problemas difíceis são, de certa forma, equivalentes. Esta teoria é frequentemente utilizada na identificação de problemas para os quais não se espera a obtenção de algoritmos de tempo polinomial para sua resolução. No entanto, diversos problemas NP-completos e NP-difíceis ainda devem ser solucionados na prática, e portanto é natural perguntar se cada um desses problemas admite algoritmos cuja complexidade de tempo não polinomial seja uma função puramente de alguns aspectos do problema. Questões sobre a existência de tais algoritmos são tratados no âmbito da Teoria da Complexidade Parametrizada desenvolvida por Downey e Fellows [13, 14, 20].

A Teoria da Complexidade Parametrizada surgiu como uma alternativa promissora para se trabalhar com problemas NP-difíceis. Vários problemas de difícil solução podem ser descritos da seguinte forma: "dado um objeto x e um inteiro não negativo k, x tem alguma propriedade que depende de k?". Na Complexidade Parametrizada, k é chamado de parâmetro. O interesse em tais parâmetros se deve ao fato de que, em muitos casos, somente uma pequena faixa de valores é realmente importante na prática. Logo, o parâmetro k pode ser considerado pequeno em comparação com o tamanho de x. Sendo assim, a intratabilidade (aparente) desses problemas no caso geral pode ser indevidamente pessimista. Desta forma, analisando-se mais profundamente a estrutura da entrada (considerando-a com um conjunto de parâmetros adicionais), deseja-se saber se existem algoritmos determinísticos que são exponenciais somente com relação a k mas polinomiais com relação a x.

Por exemplo, muitas vezes determinar o custo mínimo para a produção de uma empresa, ou o lucro máximo que esta poderá obter, são problemas difíceis de serem resolvidos. No entanto, na prática, as empresas geralmente necessitam determinar apenas se é possível efetuar a produção com um determinado orçamento ou verificar se é possível cumprir uma determinada meta de venda/lucro. Portanto, podemos adicionar aos problemas de custo mínimo ou lucro máximo destas empresas parâmetros fixos adicionais, como um orçamento ou uma meta, dando origem às suas versões parametrizadas, que podem ser mais fáceis de serem solucionadas (admitindo algoritmos de menor complexidade) e ao mesmo tempo satisfazer as necessidades requeridas.

Como podemos observar, a Teoria da Complexidade Parametrizada modela perfeitamente a realidade deste e outros cenários; muitos problemas considerados intratáveis sob o ponto de vista teórico (a menos que $P = NP$), na prática são tratáveis por parâmetro

fixo.

Além disso, um fato notável é que, embora sob o olhar convencional de NP-completude, não haja uma discriminação entre quais problemas NP-completos são mais difíceis ou mais fáceis, na prática, quando deseja-se resolver alguns problemas, é perceptível a diferença de dificuldade em suas resoluções. A complexidade parametrizada, por sua vez, captura de certa forma estas diferenças de dificuldade entre problemas NP-completos.

Uma outra maneira de se observar a importância de algoritmos FPT, decorre do fato de que seu uso fornece uma análise mais refinada da complexidade dos algoritmos. No desenvolvimento e análise de algoritmos, em geral descreve-se a complexidade dos problemas em termos do tamanho da entrada. Entretanto, é comum encontrarmos algoritmos com a mesma complexidade e que frequentemente possuem tempo de execução muito distintos, por dependerem de características específicas da instância que está sendo resolvida, e não apenas de seu tamanho.

A importância dos algoritmos FPT pode ser notada também pelo fato de que diversos algoritmos presentes na literatura, e desenvolvidos antes do surgimento desta teoria, são de fato algoritmos FPT, como o algoritmo de Lenstra para programação inteira [18].

Diversos problemas combinatórios têm sido estudados através da Teoria da Complexidade Parametrizada, como, por exemplo: COBERTURA POR VÉRTICES – provado tratável por parâmetro fixo com um parâmetro k para o tamanho da cobertura [13]; CORTE MÁXIMO – provado ser tratável por parâmetro fixo com um parâmetro k para o tamanho do corte; CLIQUE – provado ser W[1]-completo, onde k é um parâmetro para o tamanho da clique [13]; e CONJUNTO DOMINANTE – provado ser W[2]-completo com um parâmetro k para o tamanho do conjunto [13]. As classes de complexidade W[1] e W[2] são classes de intratabilidade parametrizadas, e serão descritas formalmente mais adiante no texto. Informalmente, estas classes podem ser interpretadas com classes de problemas para os quais não se acredita ser possível o desenvolvimento de algoritmos FPT. Diversos outros resultados podem ser consultados em [13, 14, 20].

Este texto introdutório não se propõe a abordar todos os aspectos da complexidade parametrizada, mas sim fornecer um visão geral da área, através da exposição do leitor aos principais conceitos e com exemplos ilustrativos das principais técnicas. Nosso objetivo é proporcionar um primeiro contato com esta rica e recente área de pesquisa, fornecendo ao leitor interessado referências para aprofundamento posterior.

5.2. Preliminares

Um *problema computacional* é uma questão a ser respondida, tipicamente contendo diversas variáveis cujo valores são não especificados. Uma *instância* de um problema é a especificação de valores para suas variáveis. A descrição de um problema é então dada pela especificação de suas instâncias e a natureza das soluções destas instâncias.

Um *problema de decisão* Π consiste de um conjunto D_Π de instâncias e um conjunto $Y_\Pi \subseteq D_\Pi$ de *instâncias sim*. Um problema de decisão é descrito informalmente pela especificação de: (i) uma instância genérica em termos de suas variáveis; (ii) uma *questão sim-não* declaradas em termos da instância genérica.

Um *problema de otimização* Π consiste de um conjunto D_Π de instâncias, uma função objetivo g e um conjunto S_Π de soluções tais que para cada $I \in D_\Pi$, existe um conjunto associado $S_\Pi[I] \subseteq S_\Pi$ de soluções para I. Um problema de otimização é descrito informalmente especificando: (i) uma instância genérica em termos de suas variáveis; (ii) a função objetivo g a ser calculada, e as propriedades que devem ser satisfeitas por qualquer solução associada a uma instância. Uma solução ótima $S_\Pi[I]$ é a solução que maximiza/minimiza o valor $g(S_\Pi[I])$.

Um *algoritmo A* para um problema Π é uma sequência finita de instruções para um computador, com a finalidade de solucionar Π. Um *algoritmo de tempo polinomial* é definido como um algoritmo cuja sua função de complexidade de tempo é $O(p(n))$, para alguma função polinomial p, onde n é usado para denotar o tamanho da entrada [16].

Definição 1 *Um problema* Π *pertence à classe* P *se e somente se* Π *pode ser solucionado em tempo polinomial por um algoritmo determinístico.*

Definição 2 *Um problema* Π *pertence à classe* NP *se e somente se para um dado certificado (uma string que certifica a resposta de uma computação), há um algoritmo determinístico que verifica sua validade em tempo polinomial.*

Definição 3 *Dados dois problemas* Π *e* Π', $\Pi \propto \Pi'$ *(*Π *se reduz a* Π' *em tempo polinomial) se existe um algoritmo que, dado uma instância I de* Π*, constrói uma instância* I' *de* Π' *em tempo polinomial em* $|I|$ *tal que a partir de uma solução para* I'*, uma resposta correta para I pode ser emitida em tempo polinomial, e vice-versa.*

Definição 4 *Um problema* Π' *é* NP-difícil *se para todo problema* $\Pi \in NP$, $\Pi \propto \Pi'$*; se* Π' *também está em NP, então* Π' *é NP-completo.*

É fácil ver que $\Pi \in P$ implica em $\Pi \in NP$. Se um único problema NP-difícil pode ser solucionado em tempo polinomial, então todo problema em NP pode ser solucionado em tempo polinomial. Se qualquer problema em NP não pode ser solucionado em tempo polinomial, então nenhum outro problema NP-completo poderá ser solucionado em tempo polinomial. Um problema NP-completo Π, portanto, possui a seguinte propriedade: $\Pi \in P$ se e somente se $P = NP$. A questão "$P = NP$?" é a mais importante questão em aberto da ciência da computação.

Um algoritmo é dito *eficiente* se sua complexidade satisfaz algum critério, por exemplo, a complexidade é polinomial no tamanho da entrada. Um problema é dito *tratável* se admite um algoritmo eficiente; caso contrário, o problema é dito ser *intratável*. Como existem diveros critérios que podem ser usados para definir eficiência, há diversos possíveis tipos de tratabilidade e intratabilidade [21].

5.2.1. Tratabilidade Parametrizada na Prática

A teoria da NP-completude foi desenvolvida para mostrar quais problemas provavelmente não admitem algoritmo de tempo polinomial. Desde o início desta teoria em 1971, milhares de problemas têm sido mostrados ser NP-difíceis e NP-completos.

Embora seja interessante saber quais problemas não admitem algoritmos polinomiais, a menos que $P = NP$, um fato inconveniente permanece: esses problemas (especialmente aqueles com aplicações no mundo real) ainda devem ser solucionados. Assim, a seguinte questão emerge:

"Como solucionar um problema NP-difícil da forma mais eficiente possível na prática?"

Primeiramente, temos duas possibilidades:

× - Tentar construir um algoritmo de tempo polinomial (implica $P = NP$).

√ - Invocar algum tipo de algoritmo heurístico ou aproximativo, de tempo polinomial.

Entretanto, caso estejamos restritos a soluções exatas e ótimas, heurísticas e aproximações podem não ser suficientes. Frequentemente, problemas práticos possuem diversos aspectos adicionais que podem ser considerados como, por exemplo, o grafo de entrada ser planar, as instâncias possuírem grau máximo ou diâmetro constantes, a estrutura buscada possuir tamanho limitado, e assim por diante.

Sendo assim, dado que na prática alguns aspectos do problema possuem tamanho ou valor delimitado, existem outras abordagens para a sua resolução:

× - Invocar algum tipo de técnica de"força bruta", que pode ser executada em tempo polinomial com complexidade $O(n^{f(k)})$, onde n é usado para denotar o tamanho da entrada e k é algum aspecto com tamanho ou valor delimitado. Neste caso, quando as instâncias a serem resolvidas são grandes, esta abordagem pode não ser muito viável.

√ - Invocar um algoritmo de tempo não polinomial tal que sua complexidade de tempo não polinomial é *apenas* uma função de algum subconjunto de aspectos do problema que possuem tamanho ou valor delimitado na prática.

Como podemos observar, um problema genérico possui inúmeros aspectos que poderiam ser considerados para análise. No entanto, para cada aplicação existe um subconjunto particular de aspectos do problema que são de fato interessantes. Um *parâmetro*, por sua vez, é uma função que extrai um aspecto ou um conjunto de aspectos particulares de um problema; isto é, um parâmetro é um mecânismo para isolar aspectos de um problema e um "receptáculo" no qual aspectos são encapsulados para manipulação e análise subsequente [21].

Neste ponto, as seguintes questões emergem:

1. Dado um problema e um parâmetro do problema, existe um algoritmo para o problema cuja complexidade de tempo não polinomial é puramente uma função deste parâmetro?
2. Relativo a quais parâmetros do problema tal algoritmo existe?

Se um problema Π para um conjunto K de parâmetros admite um algoritmo como descrito em (1), i.e. executável em tempo $f(K).n^{O(1)}$, então $\Pi \in FPT$ com relação a K (a classe de problemas *tratáveis por parâmetro fixo*). Alternativamente, pode ser possível demonstrar que tal algoritmo provavelmente não existe, estabelecendo uma intratabilidade desta versão do problema. Ao longo do texto, usaremos o termo FPT para nos referirmos tanto à classe dos problemas tratáveis por parâmetro fixo quanto também aos algoritmos que possuem complexidade da forma $f(K).n^{O(1)}$. Assim, podemos dizer que um problema está na classe FPT caso admita um algoritmo FPT.

5.3. Definições

Em contraste com a definição clássica de problema, um *problema parametrizado* consiste de uma primeira componente que pode ser pensada como um problema de decisão ou otimização convencional. Já a segunda componente é o *parâmetro*.

Como podemos observar, na teoria de complexidade parametrizada, estamos interessados em problemas especificados por três itens: (i) a instância de entrada; (ii) o parâmetro a ser analisado; (iii) a questão a ser respondida.

Definição 5 *Um problema parametrizado Π é descrito informalmente pela especificação de:*

- *Uma instância genérica em termos de suas variáveis.*
- *Os aspectos do problema que constituem os parâmetros.*
- *Uma questão a ser respondida em termos da instância genérica.*

Definição 6 *Seja Π um problema NP-difícil e seja $S = \{a_1, a_2, \ldots, a_\ell\}$ um subconjunto de aspectos de Π. Denotamos por:*

- *$\Pi(a_1, a_2, \ldots, a_\ell)$, ou $\Pi(S)$, a versão parametrizada de Π onde os aspectos em S são parâmetros fixados.*

Definição 7 *[13] Um problema parametrizado $\Pi(S)$ pertence à classe XP se existe um algoritmo para solucionar $\Pi(S)$ em tempo $f(S).n^{g(S)}$, onde n é usado para denotar o tamanho da entrada e f e g são funções arbitrárias.*

Lema 1 *Dado um problema NP-difícil Π e um subconjunto S de seus aspectos, se Π permanece NP-difícil mesmo quando os aspectos em S são delimitados por constantes, então um problema parametrizado $\Pi(S)$ não está em XP, a menos que $P = NP$.*

Prova. Se Π está em XP então por definição este problema é solucionado por um algoritmo que pode ser executado em tempo $f(S)n^{g(S)}$ para algumas funções f e g. Quando o valor de todos os aspectos em S são delimitados por uma constate, os valores de $f(S)$ e $g(S)$ são constantes e este tempo de execução torna-se polinomial em n. Como Π é NP-difícil então $P = NP$.

Corolário 2 *Se $P \neq NP$, então $\Pi(S)$ pertence a XP se e somente se Π é solucionável em tempo polinomial quando os aspectos em S são delimitados por constantes.*

Da mesma maneira que a noção de *tempo polinomial* é central para a formulação clássica de complexidade computacional, a noção central para a complexidade parametrizada é a *tratabilidade por parâmetro fixo*.

Definição 8 *Um problema parametrizado $\Pi(S)$ pertence à classe FPT, ou é* tratável por parâmetro fixo*, se existe um algoritmo para solucionar $\Pi(S)$ em tempo $f(S).n^c$, onde n denota o tamanho da entrada, c é uma constante arbitrária e f é uma função qualquer.*

Dado um problema NP-difícil Π e algum subconjunto de aspectos S de Π, dizemos que S é uma *uma fonte de intratabilidade de tempo polinomial* para Π, se $\Pi(S)$ pertence a FPT.

> " Na teoria da complexidade parametrizada, o foco não está em verificar *se* um problema é difícil, a teoria parte da suposição que os problemas mais interessantes são intratáveis quando considerados classicamente. O foco desta teoria está na seguinte questão: *O que faz o problema computacionalmente difícil?* ". [13]

Nas próximas seções exibiremos técnicas para demonstrar que um problema é tratável por parâmetro fixo, e em seguida discutiremos a teoria relacionada à intratabilidade parametrizada de certos problemas.

5.4. Árvores de busca de altura limitada

Diversos algoritmos para problemas combinatórios são recursivos. Estes algoritmos, tipicamente marcam, removem ou rotulam elementos de algum conjunto ou estrutura, como um grafo ou uma *string* a cada chamada recursiva. Normalmente, estes algoritmos vão reduzindo o tamanho da instância até esta ter uma resposta facilmente computável ou até mesmo trivial. Em geral, uma forma simples de se analisar a complexidade de algoritmos desta natureza consiste em calcular a quantidade de chamadas à função recursiva e multiplicar esta quantidade pela complexidade das operações efetuadas dentro de cada chamada.

Pode-se construir uma árvore enraizada para representar as chamadas recursivas: a chamada inicial à função é representada pelo nó raiz e, para cada chamada de uma instância de um procedimento a si mesmo, cria-se um novo nó como filho do nó atual. Esta estrutura das chamadas recursivas pode ser particularmente interessante para se fazer a análise dos algoritmos pois, como pode-se verificar, se todo nó interno da árvore possuir pelo menos 2 filhos, então temos que o número de nós internos é menor que o número de folhas. Assim, para determinarmos um limite para o número total de nós internos, basta calcular o número de folhas, o que é tipicamente mais fácil.

Algoritmos recursivos são particularmente úteis para a resolução de problemas por força bruta. Se, em algum problema, devemos fazer uma escolha dentre k opções,

uma possibilidade é efetuar k chamadas recursivas, com as instâncias modificadas com cada uma destas escolhas. Por exemplo, se desejamos selecionar um subconjunto de vértices de um grafo com determinada estrutura, temos, para cada vértice, duas opções: ele pertence ou não ao conjunto. Esta abordagem é bastante comum na resolução ingênua de problemas NP-Difíceis e, para instâncias pequenas, podem ser bastante eficientes. Entretanto, o fator de ramificação da árvore, ou seja, o número de filhos de um nó, depende do número de escolhas disponíveis. Se cada nó possui k filhos e a árvore possui n níveis, é fácil verificar que o número de folhas e também, portanto, o número de nós da árvore é da ordem de $O(k^n)$. Se n é o tamanho da instância, este tipo de algoritmo não é um algoritmo FPT. Por outro lado, se tanto o fator de ramificação quanto a altura da árvore são limitados por uma função de um parâmetro, tal algoritmo nos fornece naturalmente um algoritmo FPT, como veremos a seguir. Algoritmos cuja execução corresponde a árvores desta natureza são o assunto desta seção.

Além da forma mais simples apresentada acima, é comum que, em algoritmos mais sofisticados, o número de filhos de um nó e as alturas das folhas não sejam constantes. Em particular, é comum que ramos diferentes da árvore, isto é, subárvores diferentes, possuam tamanhos distintos. Estes fatores, dependem das chamadas recursivas dos algoritmos e de seus parâmetros. Embora de análise mais complexa, diversos algoritmos se utilizam destas assimetrias para a obtenção de tempos de execução mais eficientes. Veremos como utilizar este tipo de artifício nos exemplos a seguir.

Apresentaremos a seguir nosso primeiro algoritmo FPT e, a seguir, verificaremos que, utilizando-se a estrutura do problema, podemos desenvolver algoritmos mais eficientes.

Consideraremos o problema COBERTURA POR VÉRTICES (em inglês, *vertex cover* ou VC). Seja G um grafo e S um subconjunto de $V(G)$. Dizemos que S é uma cobertura por vértices se, para cada aresta e de $E(G)$, pelo menos uma das extremidades de e pertence a S. De forma equivalente, $G \setminus S$ não contém nenhuma aresta. No problema COBERTURA POR VÉRTICES, estamos interessados em encontrar o menor conjunto com esta propriedade, ou seja, uma cobertura por vértices mínima. Em sua versão de decisão, além do grafo G é fornecido também um inteiro k e deseja-se determinar se existe um conjunto de cobertura com no máximo k vértices. A seguir apresentamos a definição formal da versão parametrizada clássica do problema COBERTURA POR VÉRTICES .

Cobertura por Vértices(k)
Instância: Um grafo $G = (V,E)$.
Parâmetro: Um inteiro positivo k.
Questão: G possui um conjunto de vértices I, tal que $|I| \leq k$ e toda aresta de G possui pelo menos um de seus extremos em I?

O problema COBERTURA POR VÉRTICES é um dos problemas mais clássicos de teoria de grafos e da computação e foi um dos 21 problemas NP-completos estudados no clássico artigo de Karp [17]. Sua importância pode ser verificada pela naturalidade com que problemas reais podem ser modelados como o problema de cobertura. Por exemplo, considere um grafo que modele conflitos, e uma aresta pode representar uma briga entre indivíduos ou tarefas que não podem ser realizadas simutaneamente. Neste caso, COBERTURA POR VÉRTICES consiste em determinar o menor conjunto de vértices cuja remoção

elimina todos os conflitos.

5.4.1. Um primeiro algoritmo para o problema COBERTURA POR VÉRTICES

Da definição do problema, sabemos que em qualquer cobertura por vértices S, cada aresta deve possuir pelo menos uma de suas extremidades em S. Por outro lado, ao inserirmos um vértice v em S, sabemos que todas as arestas adjacentes a ele estão satisfeitas e, portanto, elas não são mais importantes para o grafo. Assim, tanto o vértice inserido em S quanto as arestas que o incidem podem ser removidas do grafo, sem perdermos nenhuma informação relevante. Tendo isso em mente, uma ideia de algoritmo consistiria de escolher uma aresta uv arbitrariamente e efetuar chamadas recursivas com as duas escolhas possíveis: a inserção de u ou de v em S. Note que, como o nosso parâmetro é o tamanho da cobertura, ao inserirmos um elemento em S, só podemos selecionar outros $k-1$ elementos de G para completar a cobertura por vértices. Em outras palavras, se possuíamos uma instância (G,k), faremos chamadas recursivas para $(G-u,\ k-1)$ e $(G-v,\ k-1)$. Note que se o nosso parâmetro em alguma chamada for igual a zero, e o grafo possuir alguma aresta, então sabemos que não há solução possível. Por outro lado, se em algum momento não houver mais arestas em G, sabemos que os elementos já inseridos em S são suficientes para cobrir as arestas do grafo. Assim, se nossa instância chegar a alguma dessas situações, sabemos que não há mais chamadas a serem feitas. Como, a cada chamada recursiva o número de arestas e o tamanho do parâmetro diminuem, é fácil ver que o algoritmo para. Como ele explora todas as situações possíveis, é também fácil verificar sua corretude.

De forma mais estruturada, o algoritmo pode ser escrito como a seguir.

Algorithm 1: Primeiro algoritmo para cobertura por vértices.

```
1  CoberturaDeVertices (G, k):
2  se G não possui arestas então
3  |  retorna SIM
4  fim
5  se k = 0 então
6  |  retorna NÃO
7  fim
8  Escolha uma aresta uv arbitrariamente
9  se CoberturaDeVertices (G − {u}, k − 1) retornar SIM então
10 |  retorna SIM
11 fim
12 se CoberturaDeVertices (G − {v}, k − 1) retornar SIM então
13 |  retorna SIM
14 fim
15 retorna NÃO
```

Note que, no pior caso, o algoritmo efetua duas chamadas recursivas, tantas vezes quanto possível. Em outras palavras, no pior caso podemos assumir que nenhum nó que potencialmente pudesse ter dois filhos não fará chamadas recursivas. Por outro lado,

como o nosso parâmetro sempre diminui de uma unidade a cada nível, nunca teremos uma altura maior que o parâmetro k inicial. Assim, o número de folhas será no máximo 2^k e, portanto, temos um algoritmo de parâmetro fixo para o problema COBERTURA POR VÉRTICES parametrizado pelo tamanho da cobertura.

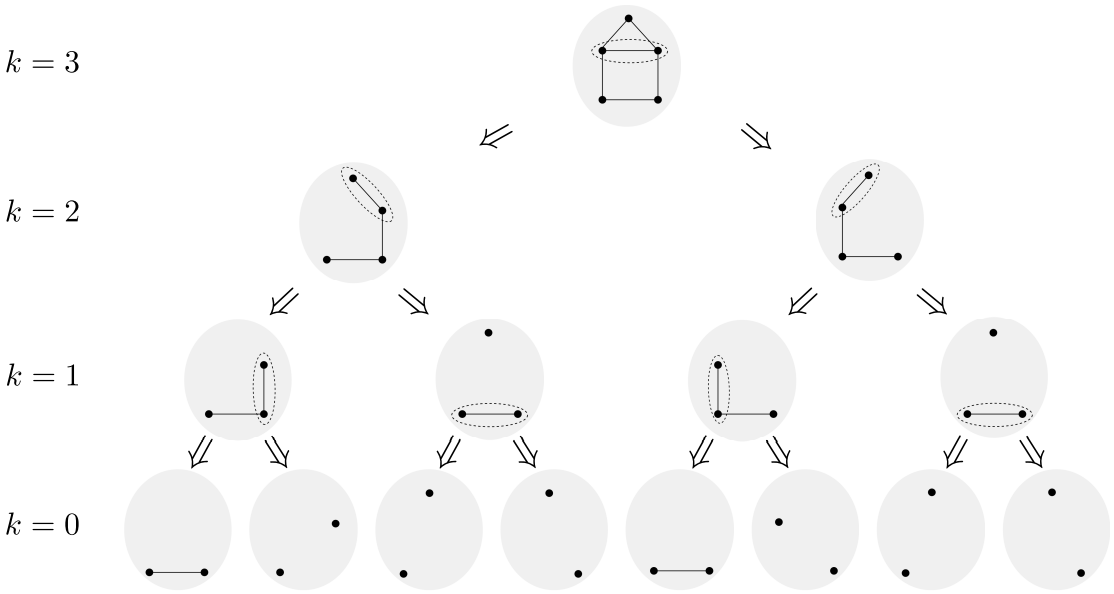

Figure 5.1. Execução do algoritmo para o grafo desenhado no topo, com $k = 3$. Observe que a cada chamada recursiva, duas novas chamadas são feitas, uma para cada extremidade da aresta selecionada. Note que algumas das instâncias do ultimo nível não possuem arestas, portanto em qualquer destes casos o algoritmo retornaria SIM.

5.4.1.1. Utilizando a estrutura do problema para a obtenção de algoritmos mais eficientes

É comum a obtenção de algoritmos mais eficientes através da análise da estrutura dos problemas. Em particular, no caso de grafos, sua estrutura combinatória frequentemente fornece ferramentas para a melhoria dos problemas. Considere o vértice v de maior grau de uma instância de VC. Lembre-se que, o grau de um vértice é o número de arestas incidentes àquele vértice. É fácil verificar que, caso o maior grau, denotado por $\Delta(G)$, seja 1, então o nosso grafo é um conjunto de arestas não adjacentes e possivelmente vértices isolados. Assim, qualquer cobertura por vértices ótima terá exatamente uma extremidade de cada aresta e, portanto, podemos escolher arbitrariamente estas extremidades.

Vamos assumir, então, que o grau de v é pelo menos 2. Note que se v não fizer parte da nossa cobertura por vértice, todos os vértices de $N(v)$, isto é, todos os vizinhos de v, farão parte da cobertura, pois, em caso contrário, alguma aresta estaria descoberta. Assim, em nosso algoritmo, ao invés de remover uma das duas extremidades de uma aresta em cada chamada recursiva, podemos remover um vértice ou todos os seus vizinhos. De fato, é possível verificar que o número máximo de folhas é reduzido neste, caso. Para de-

terminar esta quantidade vamos, primeiramente, definir $T(n)$ como o número máximo de folhas de uma chamada ao nosso procedimento quando o parâmetro é n. É fácil verificar que

$$T(k) = T(k-1) + T(k-d(v)).$$

Por outro lado, como assumimos $d(v) \geq 2$, temos

$$T(k) \leq T(k-1) + T(k-2).$$

Como podemos resolver o problema em tempo constante nos casos em que $k \leq 1$, temos $T(k) = 1$ nestes casos. Mostraremos a seguir que a complexidade deste algoritmo é menor que a versão anterior.

Teorema 3 COBERTURA POR VÉRTICES *pode ser resolvido em tempo* $O(n^{O(1)}1,6181^k)$.

Prova: Primeiramente, observe que o número de passos em uma chamada recursiva é polinomial em n, portanto, $O(n^c)$ para alguma constante c. Resta mostrar que $T(k) \leq 1,6181^k$. Para isso, usaremos indução. Observe que os casos $k = 0$ e $k = 1$ são claramente verdadeiros. Seja $k \geq 2$. Então, temos $T(k) \leq T(k-1) + T(k-2)$ e, pela hipótese de indução, $T(k) \leq 1,6181^{k-1} + 1,6181^{k-2} \leq 1,6181^{k-2}(1 + 1,6181) \leq 1,6181^k$, onde a última desigualdade é válida pois $1 + 1,6181 \leq 1,6181^2$, o que pode ser facilmente verificado.

5.4.1.2. Explorando ainda mais a estrutura dos grafos

Como acabamos de verificar, é possível utilizar a estrutura dos grafos para desenvolver algoritmos mais eficientes. De fato, exploraremos o raciocínio acima uma vez mais, desenvolvendo um terceiro algoritmo para COBERTURA POR VÉRTICES.

Para obtermos um algoritmo eficiente, utilizaremos o seguinte resultado.

Teorema 4 COBERTURA POR VÉRTICES *pode ser resolvido em tempo polinomial se* $\Delta(G) \leq 2$.

De fato, grafos com grau máximo 2 são a união de ciclos, caminhos e vértices isolados e uma cobertura por vértice para um grafo com esta estrutura pode ser obtida a partir da união das coberturas de vértices das partes que o compõem.

Assim, como no caso acima o algoritmo só fazia chamadas recursivas com vértices de grau pelo menos 2, agora estas chamadas só ocorrerão para vértices com grau pelo menos três. Assim, nossa relação de recorrência para o número de nós também é alterada, passando a ser $T(k) \leq T(k-1) + T(k-3)$ para $k \geq 3$ e $T(k) = 1$, nos demais casos. Usando técnicas similares às anteriores é fácil demonstrar o seguinte resultado:

Teorema 5 COBERTURA POR VÉRTICES *pode ser resolvido em tempo* $O(n^{O(1)}1,4656^k)$.

Algorithm 2: Segundo algoritmo para cobertura por vértices.

```
1  CoberturaDeVertices (G, k):
2  se Δ(G) < 2 então
3  |  se |E(G)| ≤ k então
4  |  |  retorna SIM
5  |  senão
6  |  |  retorna NÃO
7  |  fim
8  fim
9  se k = 0 então
10 |  retorna NÃO
11 fim
12 Seja v um vértice de grau Δ(G)
13 se CoberturaDeVertices (G − {v}, k − 1) retornar SIM então
14 |  retorna SIM
15 fim
16 se Δ(G) ≤ k então
17 |  se CoberturaDeVertices (G − {N(v)}, k − Δ(G)) retornar SIM
   |  então
18 |  |  retorna SIM
19 |  fim
20 fim
21 retorna NÃO
```

Algorithm 3: Terceiro algoritmo para cobertura por vértices.

```
1  CoberturaDeVertices (G, k):
2  se Δ(G) ≤ 2 então
3  |  se Se existe cobertura de tamanho ≤ k então
4  |  |  retorna SIM
5  |  senão
6  |  |  retorna NÃO
7  |  fim
8  fim
9  se k = 0 então
10 |  retorna NÃO
11 fim
12 Seja v um vértice de grau Δ(G)
13 se CoberturaDeVertices (G − {v}, k − 1) retornar SIM então
14 |  retorna SIM
15 fim
16 se Δ(G) ≤ k então
17 |  se CoberturaDeVertices (G − {N(v)}, k − Δ(G)) retornar SIM
   |  então
18 |  |  retorna SIM
19 |  fim
20 fim
21 retorna NÃO
```

Embora estas reduções na base da exponencial possam parecer pequenas, observando a Figura 1.1 é possivel verificar que um número reduzido de nós seria gerado pelas chamadas recursivas.

Outra forma de verificar os benefícios da redução da complexidade é através da análise a tabela abaixo, que ilustra o número aproximado de chamadas no pior caso em cada um dos três algoritmos para valores de k iguais a 20, 30 e 40.

Algoritmo	$k = 20$	$k = 30$	$k = 40$
Algoritmo 1	1048576	1073741824	1099511627776
Algoritmo 2	15139	1862776	229199843
Algoritmo 3	2091	95601	4371377

Table 5.1. Número de chamadas à função, em cada uma das versões do algoritmo.

É importante ressaltar que, para valores maiores, estas diferenças se tornam ainda mais discrepantes. Para compreendermos melhor a ordem de grandeza em que estamos trabalhando, considere que cada chamada da função pudesse ser executada em 1 microsegundo. Nos casos em que $k = 40$, teríamos, no melhor algoritmo, um tempo de pouco mais de 4 segundos. Na versão intermediária, este tempo sobe para quase 4 minutos. Já na versão mais ingênua, este tempo chegaria a cerca de 12 dias. Assim, mesmo pequenas melhorias na complexidade podem fazer grande diferença, especialmente por se tratar da base de uma função exponencial.

5.4.2. Cadeia mais próxima

O problema da cadeia (ou *string*) mais próxima pode ser definido da seguinte forma: a partir de um conjunto de cadeias de caracteres de mesmo comprimento, desejamos encontrar uma cadeia que esteja próxima de todas, ou seja, cuja distância para as cadeias de entrada seja a menor possível. A distância entre duas cadeias, neste caso, é medida pelo número de posições nas quais estas cadeias diferem uma da outra. Mais formalmente, podemos definir o problema em sua versão de decisão da seguinte forma:

Cadeia mais próxima
Instância: Um conjunto S de k cadeias de caracteres, de comprimento ℓ e um inteiro d.
Questão: Existe uma cadeia s tal que a distância de s para qualquer elemento de S é no máximo d?

Apresentaremos um algoritmo para este problema parametrizado pela distância máxima d. Em sua versão parametrizada, o problema pode ser definido como a seguir.

Cadeia mais próxima parametrizada pela distância
Instância: Um conjunto S de k cadeias de caracteres, de comprimento ℓ.
Parâmetro: Um inteiro positivo d.
Questão: Existe uma cadeia s tal que a distância de s para qualquer elemento de S é no máximo d?

Antes de fornecer um algoritmo para este problema, apresentaremos uma observação que, embora simples, permite demonstrar que o problema é tratável por parâmetro

fixo.

Observação 1 *Se existem $x, y \in S$ tais que $dist(x,y) > 2d$, então a resposta é* NÃO.

Note que a condição acima pode ser verificada em tempo polinomial. Assim, caso haja um par de cadeias com distância maior que $2d$, podemos simplesmente responder NÃO. A partir de agora, assumiremos que não existe um par de cadeias com tal propriedade.

Um procedimento recursivo para resolver este problema pode funcionar da seguinte forma: inicialmente escolhemos arbitrariamente uma cadeia s_1 de S, e fazemos $s = s_1$. Caso a distância de s para todos os elementos de S seja no máximo d, não há mais nada o que fazer e podemos responder SIM. Caso contrário, existe uma cadeira $s_i \in S$ tal que $dist(s,s_i) > d$. Tentaremos, então, modificar posições de s, de forma que a distância em relação a s_i se torne no máximo d. Note que, tomando $d+1$ posições que tenham caracteres distintos em s e s_i, pelo menos um deles deve ser alterado, caso contrário s não terá distância no máximo d em relação a s_i. Podemos, então, testar todas essas possibilidades, com cada uma dando origem a uma chamada recursiva do procedimento. Para cada uma destas possibilidades, podemos marcar esta posição como permanentemente alterada de forma que não voltemos atrás desta decisão (caso múltiplas alterações fossem feitas na mesma posição, poderíamos efetuar apenas a última). Observe ainda que a cada alteração é efetuado um novo conjunto de chamadas recursivas e, assim, a distância de s aumenta em uma unidade em relação à cadeia inicial s_1. Como a solução final s deve satisfazer $dist(s,s_1) \leq d$, podemos ter no máximo d níveis de profundidade, totalizando $O((d+1)^d)$ chamadas recursivas.

Uma análise cuidadosa das ideias apresentadas pode fornecer um algoritmo de complexidade $(d+1)^d|S|$, como pode ser verificado em [12].

5.4.2.1. Determinando a complexidade de algoritmos recursivos

Embora uma abordagem abrangente sobre a determinação da complexidade de algoritmos recursivos esteja fora do escopo deste texto, abordaremos este aspecto de forma breve. De certa forma, as constantes obtidas nos algoritmos para o problema da cobertura por vértices podem parecer obtidas num truque de mágica mas, na verdade, existem técnicas para analisar de maneira satisfatória a complexidade de algoritmos desta natureza.

Algoritmos recursivos têm, normalmente, sua complexidade analisada através do uso de relações de recorrência. Assim, o tempo de execução de uma função é calculado em função do número de passos executados dentro da função e também do tempo de execução das chamadas recursivas, que são representados de forma implícita. Os casos base da relação de recorrência, em geral são obtidos quando os argumentos da função são, de alguma forma, pequenos (em nossos exemplos, quando o parâmetro k era demasiadamente pequeno ou quando a entrada fica simples, como no caso em que o grafo possuia alguma propriedade que o tornava "simples", como o grau máximo pequeno).

Para a resolução de relações de recorrência, recomendamos uma consulta a um texto clássico de algoritmos, como [8].

5.5. Redução a um núcleo

A forma mais natural de demonstrarmos que um problema Π é FPT, é exibindo um algoritmo de tempo $f(k).n^{\alpha}$ que solucione Π. Muitas vezes, a construção desse algoritmo não ocorre de forma imediata, sendo necessário o uso de técnicas que auxiliem a sua construção. Dentre as técnicas conhecidas para classificação de um problema como FPT, uma das mais utilizadas é o método de *redução a um núcleo* do problema.

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

De certa forma, o conceito de redução a um núcleo (em inglês, *kernelization*) pode ser pensado como a formalização da ideia de pré-processamento. Em diversos problemas é comum a possibilidade de redução do tamanho da instância através da eliminação de partes da instância que sejam trivialmente resolvíveis ou que não façam parte de solução alguma.

Como exemplo, consideremos o problema da clique máxima, que consiste em encontrar o maior conjunto de vértices tal que existe aresta entre todos os seus elementos. Assumindo que o grafo contenha alguma aresta, podemos eliminar todos os vértices isolados, pois estes certamente não farão parte da maior clique. Por outro lado, caso haja algum vértice universal, sabemos que ele fará parte de qualquer clique máxima, então podemos removê-lo e diminuir uma unidade da resposta do problema reduzido.

A técnica de redução a um núcleo consiste em efetuar tantos passos de pré-processamento quanto possível de forma que, ao final a solução seja trivial ou então que a instância seja reduzida de forma que seu tamanho seja limitado por um polinômio do parâmetro k original. Esta instância reduzida é chamada de núcleo que pode ser pensada como a parte mais difícil do problema.

Formalmente, se a instância I de um problema consiste do par (x,k), onde x é a entrada e $k \in \mathbb{N}$ é o parâmetro, o *núcleo* de I é uma instância $I' = (x',k')$ tal que $|x'| \leq g(k)$ e $k' \leq g(k)$, onde g é uma função computável qualquer, que depende apenas de k, tal que I é uma instância SIM se e somente se I' é uma instância SIM. Assim, desejamos obter uma transformação f de forma que $f(I) = I' = (x',k')$, satisfaça $I \in Y_{\Pi}$ e somente se $I' \in Y_{\Pi}$. Em particular, estamos interessados em transformações que podem ser feitas em tempo polinomial em $|x|$ e k, ou seja, cujo tempo é um polinômio que depende do tamanho da entrada e do parâmetro k. Chamamos tais reduções de *regras de redução*.

Um algoritmo de redução a um núcleo consiste, portanto, em um conjunto de regras de redução tais que, quando aplicadas à exaustão, levam uma instância $I = (x,k)$ a uma instância final $I' = (x',k')$ tal que $|x'| \leq g(k)$ e $k' \leq g(k)$, onde g é uma função computável qualquer, que depende apenas de k. Em termos mais informais, o algoritmo de redução consiste de sucessivas aplicações, tantas vezes quanto possível, de regras de redução até obtenção de uma instância final reduzida que é limitada pelo parâmetro inicial k.

O papel de um núcleo pode, à primeira vista, parecer secundário na teoria de algoritmos FPT. Entretanto o teorema a seguir nos mostra a equivalência entre a existência de um núcleo e de um algoritmo tratável por parâmetro fixo.

Teorema 6 *Um problema* Π *admite um algoritmo tratável por parâmetro fixo se e somente se ele admite um núcleo.*

De fato, como veremos na Seção 1.8, esta equivalência permite um nível mais refinado de classificação da dificuldade de problemas FPT. Mas neste momento, nos concentraremos na obtenção de núcleos para alguns problemas.

5.5.1. Um núcleo para o problema de cobertura de vértices

Como visto na seção anterior, o problema de cobertura de vértices admite algoritmos tratáveis por parâmetro fixo. Apresentaremos, agora, um algoritmo de redução a um núcleo para o mesmo problema. Para este fim, definiremos regras de redução e, em seguida, demonstraremos que, após a aplicação das regras, a instância resultante é limitada pelo tamanho do parâmetro k.

Seja (G,k) uma instância do problema de cobertura de vértices parametrizado pelo tamanho da cobertura, onde G é um grafo e k é o parâmetro. Uma primeira observação diz respeito a vértices isolados, ou seja, vértices que não possuem nenhuma aresta incidente a eles. Note que vértices isolados não farão parte de nenhuma cobertura de vértices ótima, uma vez que sua inclusão em uma cobertura não cobriria nenhuma aresta. Além disso, sua remoção do grafo também não afeta nenhuma aresta existente. Assim, podemos definir nossa primeira regra de redução.

Regra de redução 1 *Se* (G,k) *é uma instância do problema de cobertura de vértices parametrizado e o grafo G contém um vértice isolado v, então remova v de G, obtendo uma nova instância* $(G-v,k)$.

Voltemos, agora, nossa atenção a vértices de grau elevado. Seja v um vértice de grau pelo menos $k+1$, ou seja, existem pelo menos $k+1$ arestas incidentes a v. Note que, como todas estas arestas precisam ser cobertas, caso v não faça parte da cobertura de vértices, a única maneira de cobrir as arestas incidentes a ele é incluindo $N(v)$ à cobertura. Entretanto, como o grau de v é pelo menos $k+1$, precisaríamos adicionar pelo menos $k+1$ vértices, o que não é possivel, pois podemos adicionar no máximo k elementos. Assim, concluímos que v deve fazer parte de qualquer cobertura de vértices de cardinalidade no máximo k. Podemos, então, formular nossa segunda regra de redução.

Regra de redução 2 *Se* (G,k) *é uma instância do problema de cobertura de vértices parametrizado e o grafo G contém um vértice v com* $d(v) \geq k+1$*, então remova v de G e reduza uma unidade de k, obtendo uma nova instância* $(G-v,k-1)$.

Note que aplicando as regras de redução 1 e 2 tanto quanto possível, a instância resultante possuirá grau máximo k, uma vez que a existência de vértices com grau superior

a k permitiriam a aplicação da Regra de redução 2 mais vezes. Embora estas regras sejam consequências de observações simples, a adição de apenas uma terceira regra, mais técnica, nos permitirá obter o núcleo que buscamos.

Regra de redução 3 *Se $I = (G,k)$ é uma instância do problema de cobertura de vértices parametrizado tal que as Regras de redução 1 e 2 não podem ser mais aplicadas e alguma das condições abaixo é satisfeita, então I é uma instância* NÃO.

- $k < 0$, *ou*
- *G possui mais de $k^2 + k$ vértices, ou*
- *G possui mais de k^2 arestas.*

O caso em que $k < 0$ é de fácil verificação: significa que já adicionamos mais vértices ao conjunto de cobertura do que poderíamos. Para verificar a corretude das demais condições, observe que com k vértices, é possivel cobrir no máximo k^2 arestas, uma vez que o grau máximo é k. Para o limite no número de vértices, note que cada vértice tem grau pelo menos 1, uma vez que a Regra de redução 1 não pode ser mais aplicada. Assim, como todo vértice possui grau pelo menos 1 e toda aresta deve ser coberta, então cada vértice deve possuir pelo menos um vizinho em qualquer conjunto de cobertura ou fazer parte do próprio conjunto. Assim, como no conjunto de cobertura existem no máximo k vértices e cada um desses pode ter no máximo k vizinhos fora do conjunto e todos os vértices do grafo devem se encontrar em uma destas situações, temos no máximo um total de $k + k^2$ no grafo.

Podemos então concluir o seguinte resultado.

Teorema 7 *O problema de cobertura de vértices admite um núcleo com no máximo $O(k^2)$ vértices e $O(k^2)$ arestas.*

De fato, após a aplicação das regras, ou teremos concluído que a instância em questão e uma instância NÃO, através da Regra de redução 3, ou teremos uma instância cujo tamanho não poderá ser maior que $O(k^2)$. Um exemplo de aplicação das regras de redução pode ser visto na Figura 1.2

Assim como no caso das árvores de altura limitada, em que pudemos melhorar substancialmente nosso algoritmo inicial utilizando a estrutura do problema, o mesmo ocorre com os algoritmos de redução a um núcleo. Embora fora do escopo deste livro, é possível formular regras de redução de forma a obtermos um núcleo de tamanho linear para o problema [9].

5.5.2. Conjunto de arestas de retroalimentação em torneios

Um *torneio* é um grafo direcionado tal que, entre cada par de vértices u e v, existe exatamente uma aresta direcionada, (u,v) ou (v,u). Um conjunto de arestas de retroalimentação em um grafo direcionado consiste em um conjunto de arestas que, se removidas do grafo, deixam o grafo acíclico. Um grafo direcionado é acílclico se não é possivel, para nenhum

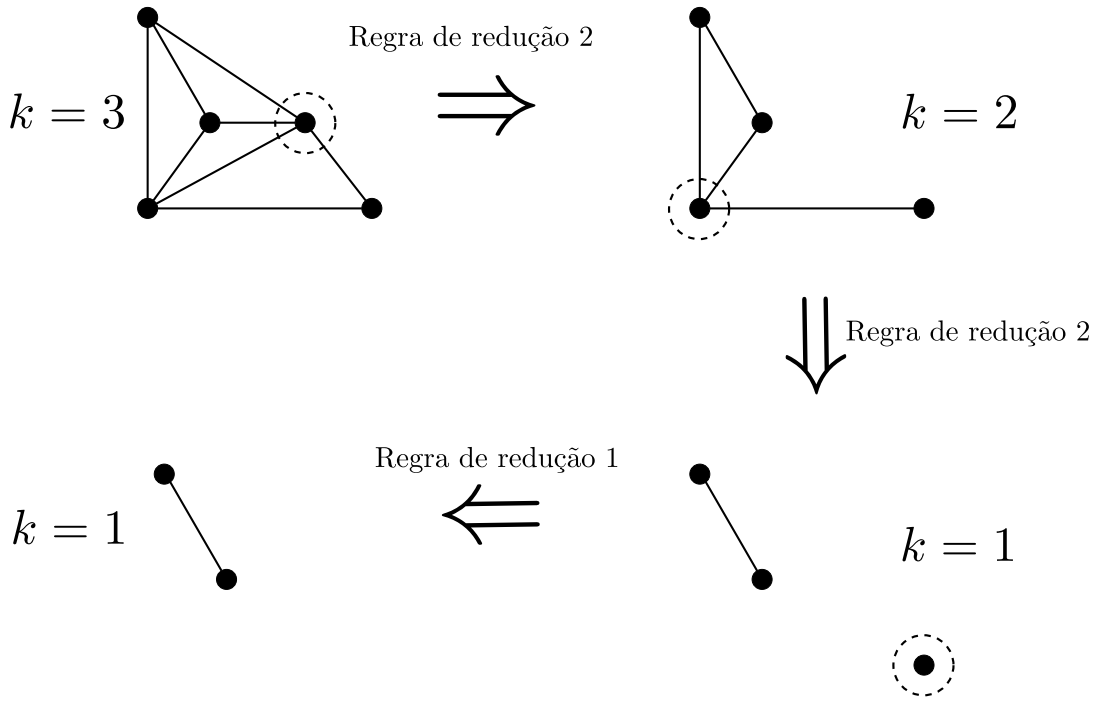

Figure 5.2. Exemplo de aplicação das regras de redução 1 e 2.

vértice inicial v, retornar a ele mesmo, após seguir uma sequência de arestas, respeitando suas direções. Nos restringiremos, aqui, ao problema do conjunto de arestas de retroalimentação no caso de torneios. Consideraremos, em particular, a versão parametrizada pelo tamanho do conjunto de retroalimentação.

Ao nos restringirmos a torneios encontramos uma dificuldade: ao remover uma aresta de um torneio, ele deixa de ser um torneio. Assim, teríamos que encontrar regras de redução que não removessem arestas do grafo original. Para contornar esta limitação técnica, exploraremos as relações entre conjuntos de arestas de retroalimentação e a reversão de arestas, ou seja, a operação que transforma uma aresta (u,v) em uma aresta (v,u). O seguinte fato pode ser verificado facilmente.

Observação 2 *Se G é um grafo direcionado e F é um subconjunto de arestas de G tal que a reversão de F deixa o grafo acíclico, então F é um conjunto de retroalimentação.*

É possível verificar que a recíprova não é verdadeira. Entretanto, alterando ligeiramente o enunciado, com a adição de condições de minimalidade, podemos obter o seguinte resultado, cuja prova deixaremos como um exercício. Recordamos que um conjunto S é minimal em relação a uma propriedade, se S satisfaz esta propriedade, mas que nenhum subconjunto próprio de S também satisfaz esta propriedade. Já um conjunto S é mínimo em relação a uma propriedade se não existe nenhum conjunto R com a mesma propriedade que satisfaça $|R| < |S|$.

Teorema 8 *Se G é um grafo direcionado e F é um subconjunto de arestas de G então F é um conjunto de retroalimentação minimal se e somente se F é um conjunto minimal cuja reversão torna o grafo acíclico.*

Utilizando o Teorema 8, podemos elaborar as regras de redução em termos de reversões de arestas, ao invés de remoção de arestas, uma vez que, se a equivalência entre os dois conjuntos vale para conjuntos minimais então, em particular, ela também vale para conjuntos mínimos.

Um triângulo, em um grafo direcionado G, é um subgrafo H de G com exatamente 3 vértices no qual todo vértice pode ser alcançado a partir de qualquer outro vértice, respeitando-se as orientações das arestas. Apresentamos, agora, as regras de redução.

Regra de redução 4 *Se (G,k) é uma instância para o problema do conjunto de arestas de retroalimentação em torneio e f é uma aresta pertencente a pelo menos $k+1$ triângulos, então podemos reverter f e diminuir k em uma unidade.*

Em outras palavras, se uma aresta faz parte de muitos triângulos, então, como devemos destruir todos os ciclos, precisaremos remover ao menos uma aresta de cada triângulo. Como a aresta f faz parte de todos eles, sua remoção destrói todos estes triângulos simultaneamente. Por outro lado, se esta aresta não fosse removida, precisaríamos de pelo menos $k+1$ remoções de arestas, sendo que só podemos realizar k remoções. Note que o raciocínio aqui é análogo ao da Regra de redução 2 do problema de cobertura de vértices.

Regra de redução 5 *Se (G,k) é uma instância para o problema do conjunto de arestas de retroalimentação em torneio e v é um vértice que não faz parte de nenhum triângulo, então podemos removê-lo de G.*

Neste caso, como o vértice v não faz parte de nenhum ciclo, v e as arestas incidentes a v são irrelevantes para o problema.

Como veremos com a regra de redução a seguir, as duas regras acima são suficientes para transformar a instância original em uma instância reduzida que forma nosso núcleo, ou ela é maior que um certo tamanho e, neste caso, podemos concluir que se trata de uma instância NÃO.

Regra de redução 6 *Se (G,k) é uma instância para o problema do conjunto de arestas de retroalimentação em torneio em que as regras anteriores não podem ser aplicadas, então, se G possui mais de $k(k+2)$ vértices, então (G,k) é uma instância* NÃO.

Para verificar a corretude da regra acima, seja (G,k) uma instância onde as Regras de redução 4 e 5 não podem ser aplicadas. Se esta é uma instância sim, então há no máximo k arestas cuja remoção elimina todos os ciclos de G. Note que cada uma dessas arestas pode fazer parte de no máximo k triângulos, caso contrário uma regra de redução

poderia ser aplicada. Assim, para cada aresta e em uma solução, além das duas extremidades de e, apenas outros k vértices podem formar ciclos com e. Por outro lado, todo vértice deve fazer parte de algum triângulo, caso contrário poderíamos aplicar outra regra de redução. Como os únicos triângulos do grafo são aqueles que passam por alguma das k arestas do conjunto de arestas retroalimentação, e cada uma pode formar triângulos com no máximo $k+2$ vértices, temos um máximo $k(k+2)$ vértices em uma instancia SIM.

Utilizando estas regras de redução, é possível concluir o seguinte resultado.

Teorema 9 *O problema do conjunto de arestas de retroalimentação em torneios possui um núcleo com no máximo k^2+2k vértices.*

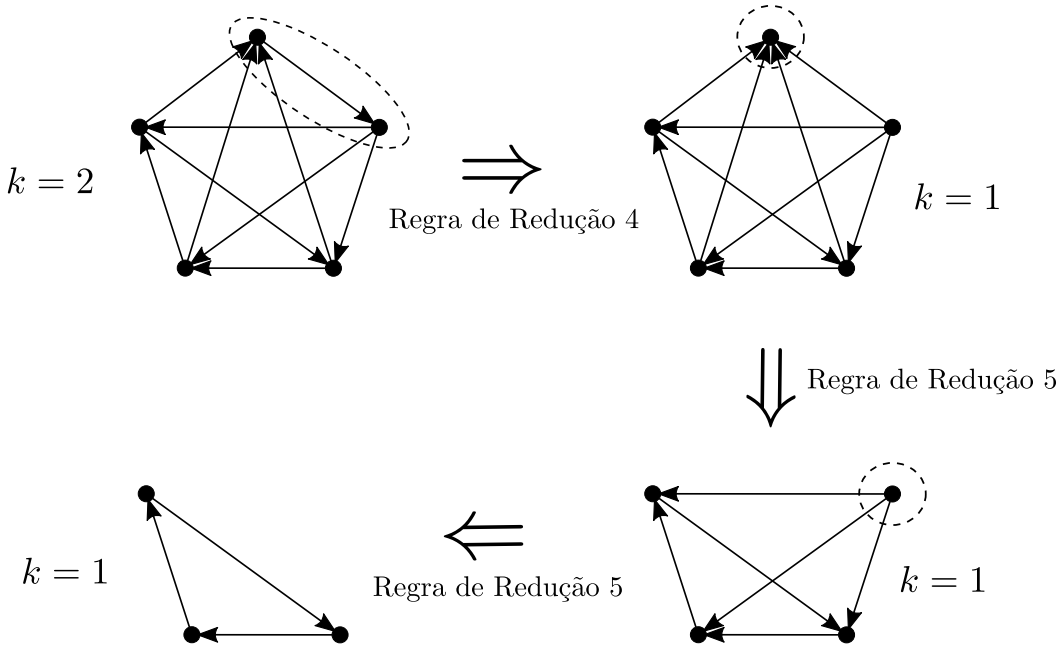

Figure 5.3. Exemplo de aplicação das regras de redução 4 e 5.

5.5.3. Outras técnicas

Assim como no caso das árvores de busca de altura limitada, a obtenção de algoritmos de redução a um núcleo dependem da estrutura do problema em questão. Entretanto, existem algumas técnicas que, embora não possam ser consideradas gerais, por se aplicarem apenas a problemas com determinada estrutura, podem ser bastante úteis na obtenção de núcleos. Como exemplo, podemos mencionar a decomposição em coroa [9], que pode ser utilizada para a obtenção de núcleos não só para problemas em grafos, como o núcleo de tamanho linear para o problema da cobertura de vértices, mas também problemas de outras naturezas, como o problema de satisfabilidade.

Uma evidência da riqueza deste tópico, é o uso de técnicas originadas em outras áreas na obtenção de núcleos, como ferramentas de otimização combinatória, álgebra linear, argumentos probabilísticos, dentre outros.

Finalmente, cabe mencionar uma noção um pouco mais geral de núcleo, denominada *núcleo de Turing*. Em linhas gerais, um núcleo de Turing pode ser visto como um procedimento que, ao invés de achar uma única instância reduzida, como vimos acima, busca a resposta para o problema através da resolução de diversas instâncias reduzidas, todas de tamanho limitado por uma função do parâmetro. Esta noção despertou interesse por permitir a decomposição de um problema em diversos problemas menores, em particular, em casos onde os núcleos tradicionais existentes se mostravam grandes demais.

5.6. Intratabilidade Parametrizada

Além da classe FPT, Downey e Fellows definiram classes apropriadas de problemas parametrizados, de acordo com seu nível de intratabilidade parametrizada. Essas classes são organizadas em uma *W-hierarquia* ($FPT \subseteq W[1] \subseteq W[2] \subseteq ... \subseteq W[P]$), e baseadas intuitivamente na complexidade dos circuitos necessários para se verificar a validade de uma solução, ou, alternativamente, na profundidade lógica natural do problema. É conjecturado que cada uma dessas classes são próprias [13, 14, 20], e se $P = NP$ então $FPT = W[P]$ [13].

Para estabelecer que um problema parametrizado é *intratável por parâmetro fixo* necessitamos das seguintes definições adicionais.

Definição 9 [14] *Seja* $\Pi(k)$ *e* $\Pi'(k')$ *dois problemas parametrizados, onde* $k' \leq g(k)$ *para alguma função computável* $g : \mathbb{N} \to \mathbb{N}$. *Uma* FPT*-redução (ou transformação paramétrica) de* $\Pi(k)$ *para* $\Pi(k')$ *é uma transformação R tal que:*

1. *Para todo x, temos que* $x \in \Pi(k)$ *se e somente se* $R(x) \in \Pi'(k')$*;*
2. *R é computável por um* FPT*-algoritmo (com relação a k);*

Se uma FPT-redução existe entre Π e Π' então Π é transformado (ou se reduz) parametricamente a Π'.

Lema 10 *(transitividade) Dado dois problemas parametrizados* Π, Π' *and* Π''*, se* Π *se reduz parametricamente a* Π' *e* Π' *se reduz parametricamente a* Π'' *então* Π *se reduz parametricamente a* Π''.

Lema 11 *(preservação da tratabilidade por parâmetro fixo) Dado dois problemas* Π *e* Π'*, se* Π *se reduz parametricamente a* Π' *e* Π' *é tratável por parâmetro fixo então* Π *é tratável por parâmetro fixo.*

As seguintes definições fornecem o análogo parametrizado da classe NP na teoria de NP-completude. Estas classes são, na maior parte, baseadas numa série de circuitos sucessivamente mais poderosos para verificação de uma solução, onde soluções são codificadas como vetores de entrada para estes circuitos e os parâmetros são codificados em pesos destes vetores de entrada [21].

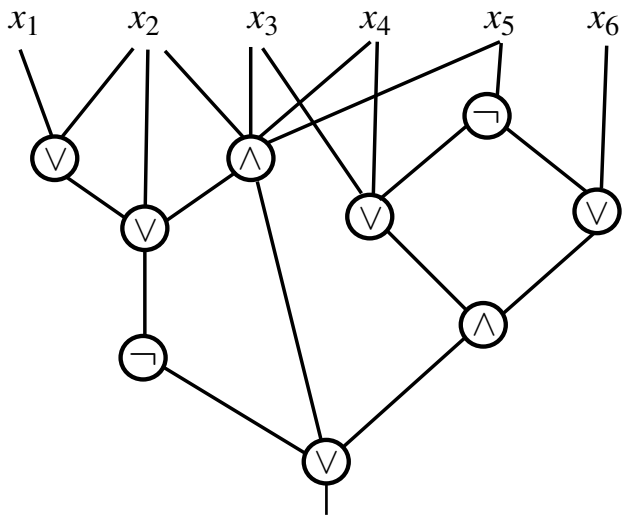

Figure 5.4. Exemplo de circuito booleano de decisão, onde (∧), (∨) e (¬) denotam portas lógicas E, Ou e de negação, respectivamente.

Um circuito booleano de decisão consiste de variáveis booleanas de entrada, portas lógicas de negação, portas lógica E, portas lógicas Ou, e uma única porta lógica de saída. Um exemplo de circuito booleano é apresentado na Figura 1.4.

O problema *Satisfabilidade de Circuito* consiste em dado um circuito booleano de decisão C, decidir se existe uma atribuição de valores às variáveis de entrada de C de tal forma que sua saída seja "verdadeiro".

Definição 10 *[13] Seja C um circuito booleano de decisão com variáveis de entrada* $x_1, \ldots, x_n$.

- *O* entrelaçamento *de C é definido como o número máximo de portas lógicas largas em qualquer caminho da variável de entrada até a linha de saída (Uma porta é denominada* larga *se suas entradas excedem algum limite constante, em geral dois).*
- *A* profundidade *de C é definida como o comprimento do maior caminho de uma variável de entrada até a linha de saída em C.*
- *O* peso *de uma atribuição às variáveis de um circuito booleano C (uma atribuição para C) é o número de* $1's$ *nesta atribuição.*

Para a próxima definição se faz necessário a seguinte formulação.

Satisfabilidade Ponderada em Circuitos de Entrelaçamento t **e Profundidade** h
WCS(t**,**h**)**
Instância: Um circuito de decisão C com entrelaçamento t e profundidade h.
Parâmetro: Um inteiro positivo k.
Questão: C possui uma atribuição satisfatível com peso k?

Definição 11 *[13] Um problema parametrizado* Π *pertence à classe W[t] se e somente se* Π *se reduz parametricamente a WCS(t,h), para alguma constante h.*

De maneira informal, um problema parametrizado Π pertence a classe $W[t]$ se existir uma redução uniforme de Π ao problema de determinar se um dado circuito lógico de decisão C aceita um vetor de entrada de tamanho k, onde C possui em qualquer caminho da entrada até a saída no máximo t portas lógicas de tamanho irrestrito e profundidade no máxima h, para alguma constante h.

Definição 12 *(A W-hierarquia) A união destas classes $W[t]$ juntamente com as classes $W[P]$ e XP, denota-se W-hierarquia. $W[P]$ denota a classe obtida por considerar nenhuma restrição sobre profundidade. Portanto, a W-hierarquia é*

$$FPT \subseteq W[1] \subseteq W[2] \subseteq \ldots \subseteq W[P] \subseteq XP.$$

Downey and Fellows conjecturaram que cada uma dessas relações de inclusão na W-hierarquia é própria [13].

Observação 3 *Uma alternativa para mostrar que um problema parametrizado Π pertence à classe W[t], $t \geq 1$, é apresentar uma FPT-redução para algum problema parametrizado pertencente a W[t]. Veja Lema 10.*

Além da W-hierarquia, na complexidade parametrizada há outras hierarquias de classes de problemas parametrizados, tais como a M-hierarquia e A-hierarquia. Entretanto, este curso terá como foco somente a W-hierarquia.

Na complexidade parametrizada, definimos $W[t]$-dificuldade e $W[t]$-completude de um problema parametrizado $\Pi(k)$ com relação à classe de complexidade $W[t]$ ($t \geq 1$), como na teoria da complexidade clássica: $\Pi(k)$ é $W[t]$-difícil sob FPT-reduções se todo problema em $W[t]$ se reduz parametricamente a $\Pi(k)$; $\Pi(k)$ é $W[t]$-completo sob FTP-reduções se $\Pi(k) \in W[t]$ e $\Pi(k)$ é $W[t]$-difícil.

5.6.1. Diferentes Níveis de Intratabilidade

Atráves da teoria da complexidade parametrizada, é possível distinguir diferentes níveis de intratabilidade entre problemas NP-completos, algo que não era possível através da complexidade clássica. Esse refinamento pode ser observado através da classificação das versões parametrizadas naturais destes problemas na W-hierarquia.

Por exemplo. Considere as naturais parametrizações dos seguintes problemas clássicos em grafos.

Conjunto Independente(c)
Instância: Um grafo $G = (V,E)$.
Parâmetro: Um inteiro positivo c.
Questão: G possui um conjunto de vértices I, tal que $|I| \geq c$ e I não contém nenhum par de vértices adjacentes?

Conjunto Dominante(k)
Instância: Um grafo $G = (V,E)$.
Parâmetro: Um inteiro positivo k.
Questão: G possui um conjunto de vértices D, tal que $|D| \leq k$ e todo vértice $v \in V \setminus D$ é adjacente a pelo menos um vértice em D?

Conjunto Independente. Conforme podemos observar na Figura 1.5, existe uma FPT-redução de Conjunto Independente(c) para WCS(1,3), onde cada vértice v_i de G é representado por uma entrada x_i, tal que toda atribuição que satisfaça o circuito associado C, induz um conjunto independente I para G, e vice-versa, sendo x_i igual a "verdadeiro" se e somente se v_i pertence ao conjunto I. Logo, pela Definição 11, temos que Conjunto Independente(c) $\in$ W[1].

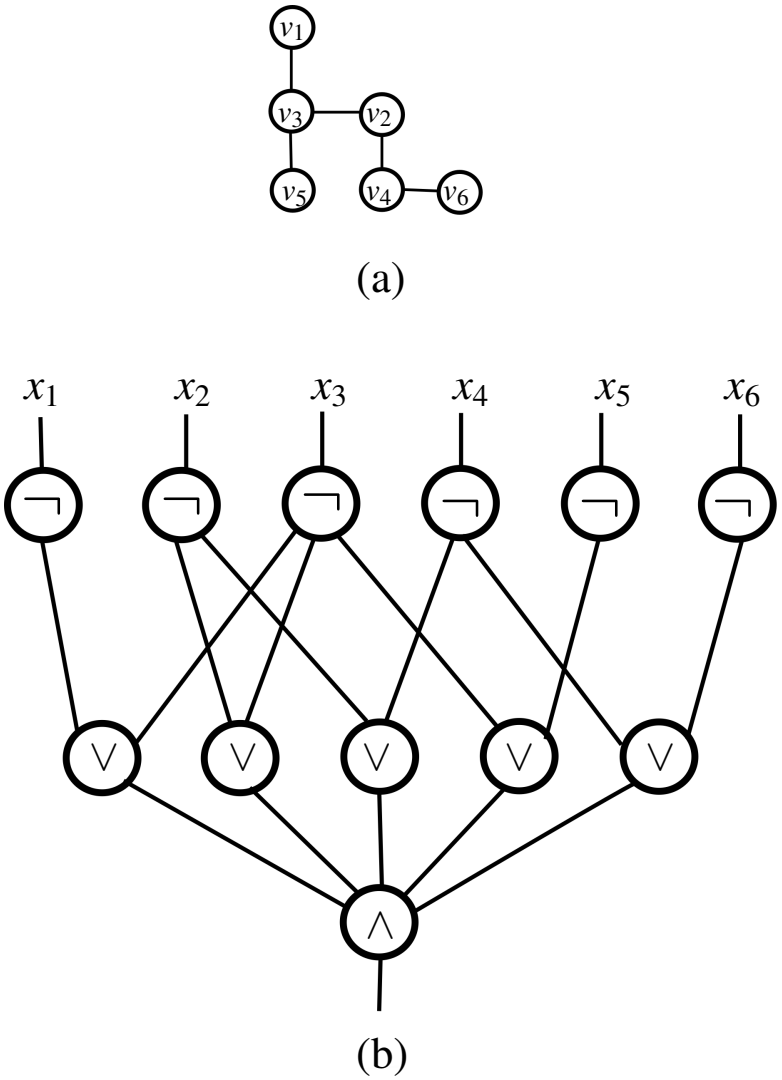

Figure 5.5. (a) Um instância G do problema Conjunto Independente(c); (b) circuito booleano de decisão C associado a G com profundidade três e apenas uma porta lógica larga em qualquer caminho a partir de uma entrada até a saída.

Conjunto Dominante. Para o problema Conjunto Dominante(k) torna-se necessário a construção de um circuito booleano um pouco mais complexo. Conforme Figura 1.6, podemos observar que existe uma FPT-redução de Conjunto Dominante(k) para WCS(2,2), onde cada vértice v_i de G é representado por uma entrada x_i, tal que toda atribuição que satisfaça o circuito associado C, induz um conjunto dominante I para G, e vice-versa, sendo x_i igual a "verdadeiro" se e somente se v_i pertence ao conjunto D. Através de tal

redução temos que Conjunto Dominante(k) $\in$ W[2].

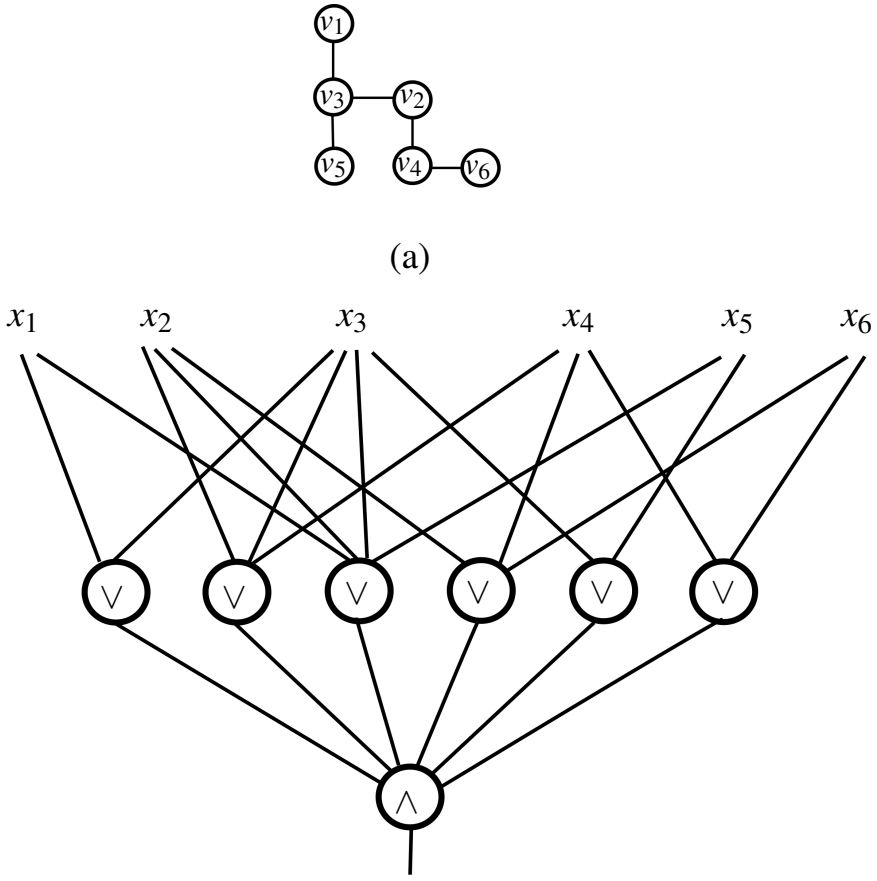

Figure 5.6. (a) Um instância G do problema Conjunto Dominante(k); (b) circuito booleano de decisão C associado a G com profundidade dois e máximo duas portas lógicas largas em qualquer caminho a partir de uma entrada até a saída.

Notem que o circuito booleano de decisão do problema Conjunto Independente(c) utiliza apenas uma porta lógica larga (entrelaçamento um), enquanto o circuito booleano de decisão do problema Conjunto Dominante(k) possui entrelaçamento dois. Poderíamos então questionar, se existe um circuito booleano de decisão para o problema Conjunto Dominante(k) com entrelaçamento menor do que dois, a resposta a esta pergunta é "provavelmente não", uma vez que Conjunto Dominante(k) é conhecido ser W[2]-completo [13], enquanto Conjunto Independente(c) é W[1]-completo [13]. Deste fato, segue que não existe uma FPT-redução do problema Conjunto Dominante(k) para o problema Conjunto Independente(c), a menos que W[1]=W[2] (o que conjectura-se não ser verdade).

O Análogo ao Teorema de Cook

Downey and Fellows [13] demonstraram um teorema análogo ao de Cook exibindo um número de problemas combinatórios que possuem complexidade parametrizada similar a versão parametrizada do problema de Aceitação da Máquina de Turing Não Determinística. Os principais problemas parametrizados são os seguintes.

Aceitação da Máquina de Turing(k)
Instância: Uma máquina de Turing não determinística $\mathbb{M}$ e uma string x.
Parâmetro: Um inteiro positivo k.
Questão: $\mathbb{M}$ aceita x com no máximo k passos?

Este problema é o análogo parametrizado do problema ACEITAÇÃO DA MAQUINA DE TURING (onde o limite no número de passos é polinomial no tamanho da entrada $|x|$, em vez de k), que é o problema NP-completo básico genérico na teoria de complexidade clássica. ACEITAÇÃO DA MÁQUINA DE TURING(k) pode ser trivialmente resolvido em tempo $O(n^{k+1})$, onde n denota o tamanho total da entrada. Isso é feito explorando-se exaustivamente todos os caminhos computacionais de k passos. Acredita-se que este resultado não possa ser significativamente melhorado.

q-CNF Satisfabilidade Ponderada(k)
Instância: Uma expressão booleana F na forma normal conjuntiva (CNF) tal que cada cláusula possui no máximo q literais ($q \geq 2$).
Parâmetro: Um inteiro positivo k.
Questão: F possui um atribuição satisfatível de peso k?

Teorema 12 *[13] (Análogo ao Teorema de Cook) Os seguintes problemas são completos para a classe W[1]:*

1. **Aceitação da Máquina de Turing**(k).
2. **q-CNF Satisfabilidade Ponderada**(k).

A partir do Teorema 12 diversos outros problemas têm sido mostrados serem W[1]-completos, tais como Clique(k) e Conjunto independente(k), onde k é o parâmetro para o tamanho dos subconjuntos buscados.

O Teorema análogo ao de Cook pode ser utilizado como ferramenta para mostrar a W[1]-dificuldade e W[1]-completude de problemas parametrizados. Pra estabelecer alguns resultados sobre classes W[t] ($t \geq 2$), Downey and Fellows apresentaram um teorema de mais alto nível, o *Teorema da Normalização*.

Definição 13 *Uma fórmula booleana F é t-normalizada se F é da forma* **produto-de-soma-de-produto...** *de literais com t-alternâncias.*

Note que a fórmula 2-normalizada é uma fórmula CNF.

Satisfabilidade Ponderada t-Normalizada(k)
Instância: Uma expressão booleana t-normalizada F $(t \geq 2)$.
Parâmetro: Um inteiro positivo k.
Questão: F possui um atribuição satisfatível de peso k?

Teorema 13 *[13] (Teorema da Normalização)* **Satisfabilidade Ponderada t-Normalizada**(k) *é completa para W[t], para todo $t \geq 2$.*

Utilizando esse último teorema muitos outros problemas parametrizados foram mostrados serem W[t]-dificil ou W[t]-completo, para algum $t \geq 2$. Por exemplo, o problema parametrizado CONJUNTO DOMINANTE(k) (k e o tamanho do conjunto buscado) foi mostrado ser W[2]-completo. Diversos outros resultados podem ser encontrados em [12, 13, 14, 20].

Como dito anteriormente, temos a relação de inclusão

$$FPT \subseteq W[1] \subseteq W[2] \subseteq ... ,$$

e existe a conjectura de que cada uma dessas relações é própria. A união das classes da W-hierarquia é denotada por WH. Se $P = NP$, temos então $WH \subseteq FPT$. A classe $W[1]$ é atualmente a classe mais estudada de problemas parametrizados.

5.7. FPT-reduções $\times$ Reduções de Tempo Polinomial

Assim como é feito com a complexidade clássica, podemos encontrar evidências para a intratabilidade parametrizada estudando as noções apropriadas de transformações de problemas. A propriedade essencial das transformações paramétricas é que, se Π pode ser transformado em Π' e $\Pi' \in FPT$, então $\Pi \in FPT$. Caso Π seja W[t]-completa, então Π' é W[t]-difícil, sendo completa para a classe $W[t]$ se $\Pi' \in W[t]$.

Nesta seção, ilustraremos alguns exemplos de FPT-reduções, e destacaremos a principal distinção entre FPT-redução e a redução de tempo polinomial (redução de Karp), a qual estamos habituados a utilizar na complexidade clássica.

Nesta seção, consideraremos os seguintes problemas:

1. COBERTURA POR VÉRTICES
2. CONJUNTO INDEPENDENTE
3. CLIQUE
4. CONJUNTO DOMINANTE

Esses quatro problemas clássicos são NP-completos, e portanto são redutíveis dois a dois através de reduções de tempo polinomial. Como exemplo, considere as seguintes reduções de tempo polinomial:

- CLIQUE $\propto$ CONJUNTO INDEPENDENTE

  Este é um dos exemplos mais simples de redução de tempo polinomial entre problemas.

  Dado uma grafo G construímos o seu grafo complementar $\bar{G}$, isto é, $V(G) = V(\bar{G})$, $E(G) \cap E(\bar{G}) = \emptyset$ e $E(G) \cup E(\bar{G}) = E(K_n)$, onde K_n é um grafo completo com $V(k_n) = V(G)$.

  Neste ponto, basta observar que G possui uma clique de tamanho k se e somente se $\bar{G}$ possui um um conjunto independente de tamanho k. A Figura 1.7 ilustra um grafo G e seu complemento $\bar{G}$, respectivamente com a clique máxima de G e o conjunto independente máximo de $\bar{G}$ destacados.

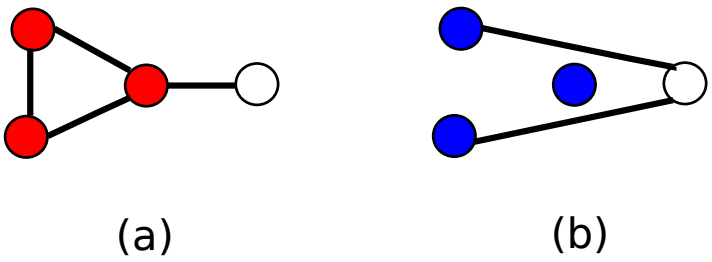

Figure 5.7. (a) Um grafo G e uma clique C de G destacada em vermelho; (b) $\bar{G}$ e o conjunto independente de $\bar{G}$ associado a C em azul.

- CONJUNTO INDEPENDENTE $\propto$ COBERTURA POR VÉRTICES

  Um outro exemplo de redução simples é apresentado a seguir.

  O problema CONJUNTO INDEPENDENTE facilmente se reduz em tempo polinomial ao problema COBERTURA POR VÉRTICES, pois dado um conjunto independente I de um grafo G, o complemento deste conjunto, $V(G) \setminus I$, é uma cobertura por vértices de G, veja Figura 1.8. Fica como exercício para o aluno verificar a corretude desta afirmação.

  Sendo assim, um grafo G possui um conjunto independente de tamanho k se e somente se G possui uma cobertura por vértices de tamanho $n-k$, onde $n = V(G)$.

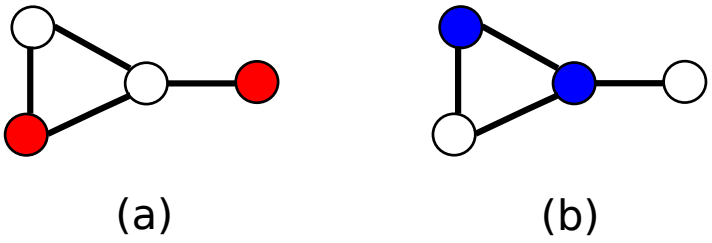

Figure 5.8. (a) Um grafo G e um conjunto independente máximo I de G destacado em vermelho; (b) G e uma cobertura por vértices mínima, $V(G) \setminus I$, destacada em azul.

- COBERTURA POR VÉRTICES $\propto$ CONJUNTO DOMINANTE

  Reduzimos COBERTURA POR VÉRTICES ao problema CONJUNTO DOMINANTE da seguinte maneira:

  Criamos um grafo G', onde inicialmente $G' = G$. Para cada aresta $e = (u,v)$ em G' adicionamos um novo vértice z_e, e arestas $(u,z_e),(v,z_e)$ formando triângulos. A Figura 1.9 ilustra um grafo G e o grafo G' associado, onde uma cobertura por vértices mínima de G e um conjunto dominante mínimo de G' encontram-se destacados.

  Neste ponto devemos provar que G possui uma cobertura por vértices de tamanho no máximo k se e somente se G' possui um conjunto dominante de tamanho no máximo k.

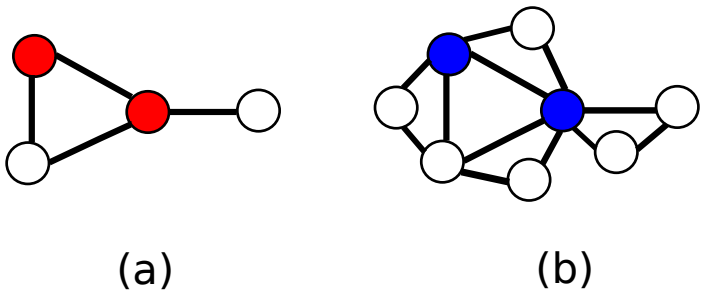

Figure 5.9. (a) Um grafo G e uma cobertura por vértices mínima de G destacada em vermelho; (b) G' e um conjunto dominante mínimo de G' destacada em azul.

Se G possui uma cobertura por vértices S, então o mesmo conjunto é um conjunto dominante de G', pois por definição de cobertura por vértice, cada aresta em G é incidente a pelo menos um vértice em S, portanto, todo vértice z_e possui pelo menos um vizinho em S.

Por outro lado, se G' possui um conjunto dominante S' então uma cobertura por vértices S de G pode ser obtida, primeiramente fazendo $S = S'$ e em seguida substituindo cada vértice $z_e \in S'$ por um de seus vizinhos em G'. Como todo vértice z_e ou pertence a S' ou possui um vizinho em S', então toda aresta de G possui pelo menos um vértice em S.

- CONJUNTO DOMINANTE$\propto$ CLIQUE

  O problema CONJUNTO DOMINANTE, por sua vez, se reduz em tempo polinomial ao problema CLIQUE da seguinte forma:

  Para cada aresta $(u,v) \in G$ criamos dois vértices $[u,v]$ e $[v,u]$ em G'. Para cada dois vértices $[u_1,v_1]$ e $[u_2,v_2]$ de G' adicionamos uma aresta entre eles se $u_1 \neq u_2$, $u_1 \neq v_2$ e $u_2 \neq v_1$. Neste ponto, devemos mostrar que G' possui uma clique de tamanho $n-k$ se e somente se G possui um conjunto dominante de tamanho k, onde $n = |V(G)|$. A Figura 1.10 ilustra um grafo G e o grafo G' associado, respectivamente com o conjunto dominante mínimo de G e a clique máxima de G' destacados.

  A intuição é que o vértice $[u,v]$ deve ser interpretado como "u é dominado por v".

  Considere um conjunto dominante S em G. Construímos um conjunto S' como segue: para cada vértice $u \notin S$ selecionamos um vizinho arbitrário v de u onde $v \in S$, e adicionamos $[u,v]$ a S'. Claramente S' é uma clique de tamanho $|V(G)| - |S|$.

  Por outro lado, considere uma clique S' em G'. Seja S o conjunto de vértices u em $V(G)$ tal que não existe vértice $v \in V(G)$ tal que $[u,v] \in S'$. Observe agora que $|S| = |V(G)| - |S'|$, dado que cada elemento de S' exclui um vértice de G. Considere um vértice $u \notin S$. Deve haver um vértice $v \in V(G)$ tal que $[u,v] \in S'$. Sabemos que $(u,v) \in E(G)$, então basta mostrar que $v \in S$. Suponha que não, então deve haver algum vértice $w \in V(G)$ tal que $[v,w] \in S'$. Mas $[u,v]$ e $[v,w]$ são não adjacentes em G', contradizendo que S' é uma clique. Portanto S é um conjunto dominante.

As reduções apresentadas acima, demonstram que os quatro problemas listados

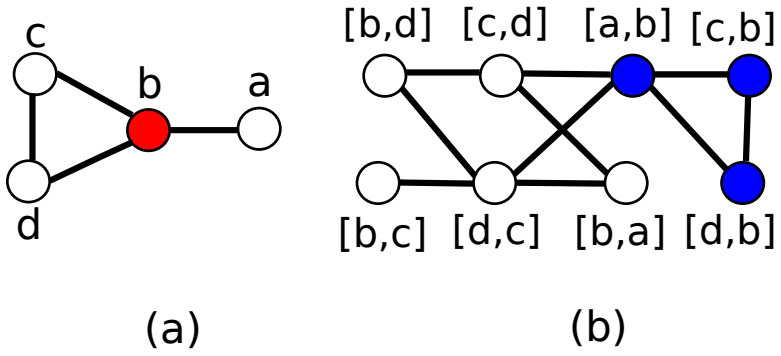

Figure 5.10. (a) Um grafo G e um conjunto dominante mínimo de G destacado em vermelho; (b) G' e uma clique máxima de G' destacada em azul.

se reduzem mutuamente através de reduções de tempo polinomial. Conforme podemos observar na Figura 1.11.

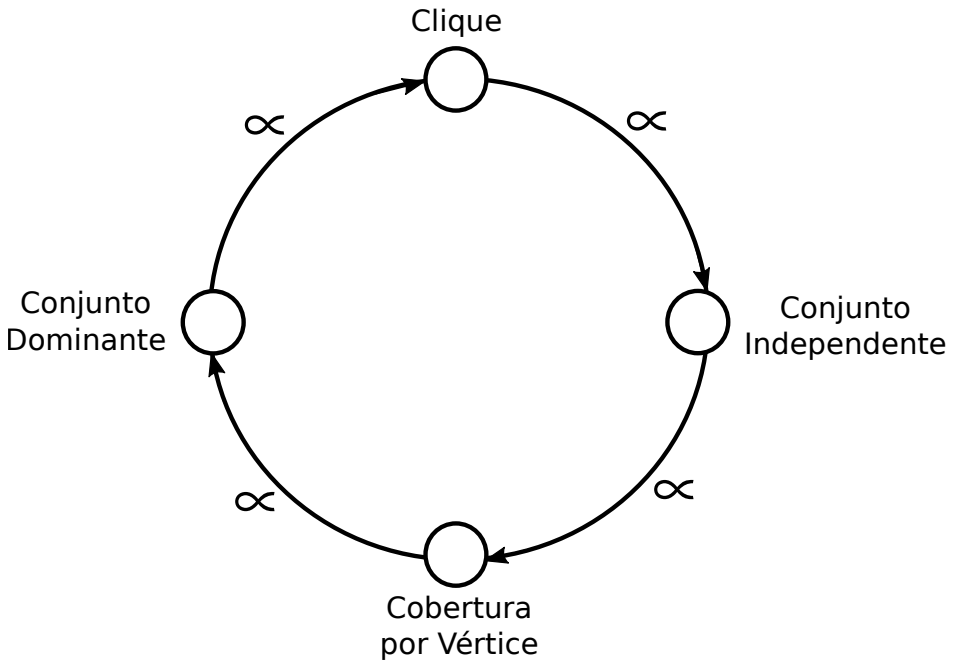

Figure 5.11. Representação das reduções de tempo polinomial apresentadas entre os problemas CLIQUE**,** CONJUNTO INDEPENDENTE**,** COBERTURA POR VÉRTICES **e** CONJUNTO DOMINANTE

Embora as reduções acima possam ser utilizadas para demonstrar a NP-dificuldade dos problemas, algumas destas reduções não podem ser utilizadas para demonstrar a intratabilidade parametrizada dos problemas (considerando como parâmetro o tamanho da estrutura buscada).

As principais características que diferem uma FPT-redução de uma redução de tempo polinomial são:

(a) FPT-reduções podem ser executadas em tempo $f(k).n^{O(1)}$ (onde k é o parâmetro e n

o tamanho da entrada), enquanto reduções de tempo polinomial, como o próprio nome diz, devem ser executadas em tempo polinomial.

Geralmente esta condição não gera dificuldades na construção de uma redução, pois tempo $f(k).n^{O(1)}$ é uma condição mais relaxada do que tempo polinomial.

(b) Em FPT-reduções, temos a restrição adicional de que o tamanho dos parâmetros do problema resultante devem ser delimitados puramente por uma função dos parâmetros do problema de origem, isto é, o tamanho dos parâmetros resultantes não podem depender do tamanho do problema original.

Esta condição é a principal diferença entre FPT-reduções e reduções de tempo polinomial. Diversas reduções de tempo polinomial não satisfazem essa condição.

Dentre as quatro reduções de tempo polinomial exibidas nesta seção, temos que:

- A redução de CLIQUE para CONJUNTO INDEPENDENTE é uma FPT-redução.
- A redução de COBERTURA POR VÉRTICES para CONJUNTO DOMINANTE é uma FPT-redução.
- A redução de CONJUNTO INDEPENDENTE para COBERTURA POR VÉRTICES *não* é uma FPT-redução, pois a cobertura por vértices resultante possui tamanho $n-k$, o que não satisfaz a condição *(b)*, uma vez que depende do tamanho da entrada.
- A redução de CONJUNTO DOMINANTE para CLIQUE também *não* é uma FPT-redução, pois a clique resultante possui tamanho $n-k$, o que não satisfaz a condição *(b)*, uma vez que depende do tamanho da entrada.

De fato, FPT-reduções de CLIQUE para CONJUNTO INDEPENDENTE e de COBERTURA POR VÉRTICES para CONJUNTO DOMINANTE eram esperadas, enquanto FPT-reduções de CONJUNTO INDEPENDENTE para COBERTURA POR VÉRTICES e de CONJUNTO DOMINANTE para CLIQUE provavelmente não devem existir pois:

COBERTURA POR VÉRTICES $\in$ FPT

CONJUNTO INDEPENDENTE é W[1]-completo

CLIQUE é W[1]-completo

CONJUNTO DOMINANTE é W[2]-completo

Sendo assim,

- Existe uma FPT-redução de CONJUNTO INDEPENDENTE ou CLIQUE para COBERTURA POR VÉRTICES se e somente se FPT=W[1].

- Existe uma FPT-redução de CONJUNTO DOMINANTE para CONJUNTO INDEPENDENTE ou CLIQUE se e somente se W[1]=W[2].

- Existe uma FPT-redução de CONJUNTO DOMINANTE para COBERTURA POR VÉRTICES se e somente se FPT=W[1]=W[2].

O leitor neste momento poderia estar se perguntando como deveriam ser as FPT-reduções de COBERTURA POR VÉRTICES para CONJUNTO INDEPENDENTE; e de CONJUNTO INDEPENDENTE para CONJUNTO DOMINANTE. Vejamos a seguir.

- COBERTURA POR VÉRTICES *se FPT-reduz a* CONJUNTO INDEPENDENTE

  Uma FPT-redução de COBERTURA POR VÉRTICES para CONJUNTO INDEPENDENTE pode ser facilmente obtida, pois COBERTURA POR VÉRTICES $\in$ FPT. Assim, podemos para cada instância G de COBERTURA POR VÉRTICES solucioná-la em tempo FPT, criando em seguida uma instância para CONJUNTO INDEPENDENTE que possui solução de tamanho k se e somente se a instância G de COBERTURA POR VÉRTICES possui solução de tamanho k.

Para ilustrar uma FPT-redução de CONJUNTO INDEPENDENTE para CONJUNTO DOMINANTE, iremos utilizar um passo intermediário. Considere o seguinte problema.

Conjunto Independente Multi-Colorido
Instância: Um grafo $G = (V,E)$ com uma k-coloração de vértices.
Parâmetro: o inteiro positivo k.
Questão: G possui um conjunto independente de vértices I contendo um vértice de cada classe de cor?

- CONJUNTO INDEPENDENTE *se FPT-reduz a* CONJUNTO INDEPENDENTE MULTI-COLORIDO.

  Dado uma instância G COBERTURA POR VÉRTICES, construímos um grafo G' criando k copias $v_1, v_2, \ldots, v_k$ para cada vértice v e colorindo v_i com cor i. Adicionamos então uma aresta em G' entre dois vértices u_i e v_j, $i \neq j$ se e somente se $u = v$ ou u e v são adjacentes em G. Neste ponto, é fácil ver que G possui um conjunto independente de tamanho k se e somente se G' possui um conjunto independente multi-colorido de tamanho k.

- CONJUNTO INDEPENDENTE MULTI-COLORIDO *se FPT-reduz a* CONJUNTO DOMINANTE.

  Seja G um grafo com seus vértices sendo k-coloridos. Construímos um grafo H tal que G possui um conjunto independente multi-colorido se e somente se H possui um conjunto dominante de tamanho k.

  Seja $(V_1, \ldots, V_k)$ os conjuntos de vértices de G, onde V_i é o conjunto de vértices coloridos com a cor i. Construímos um grafo H como segue:

  - para todo vértice $v \in V(G)$, inserimos v em H;
  - para todo $1 \leq i \leq k$, fazemos o conjunto V_i em H ser uma clique através de adição de arestas;
  - para todo $1 \leq i \leq k$, adicionamos dois novos vertices x_i, y_i em H e fazemos eles adjacentes a todo vértice em V_i;

– para toda aresta $e \in E(G)$ com extremidades $u \in V_i$ e $v \in V_j$, criamos um vértice w_e em H e o fazemos adjacente a todo vértice em $(V_i \cup V_j) \setminus \{u, v\}$.

Mostraremos que G possui um conjunto independente k-colorido se e somente se H possui um conjunto dominante de tamanho k.

Seja $Z \subset V(G)$ um conjunto independente tal que $Z = \{z_1, z_2, \ldots, z_k\}$ e $z_i \in V_i$. Agora mostraremos que Z é um conjunto dominante em H. x_i e y_i são dominados por z_i. Todos os vértices em V_i são dominados por z_i. Agora, suponha $w_{\{u,v\}}$ com $u \in V_i$ e $v \in V_j$ não possui um vizinho em Z, isto implica que $u = z_i$ e $v = z_j$, o que implica que z_i e z_j possuem uma aresta em G, o que é uma contradição.

Agora, assuma que temos um conjunto dominante D de tamanho k em H. Como todos os $x_i's$ e $y_i's$ devem ser dominados, sem perda de generalidade cada V_i deve ter pelo menos um de seus membros em D. Portanto, podemos escrever D como $D = \{d_1, d_2, \ldots, d_k\}$ tal que $d_i \in V_i$.

Nos resta mostrar que D é um conjunto independente de tamanho k de G. Claramente os vértices em D são unicamente coloridos pois pertencem a diferentes $V_i's$. Suponha que d_i e d_j possuem uma aresta entre eles em G. Isto implica que w_{d_i,d_j} não possui um vizinho em D, o que é uma contradição.

5.8. Pré-processamento e Inviabilidade de Núcleos Polinomiais

Pré-processamento (ou redução de dados) para reduzir o tamanho da instância é uma das heurísticas mais comumente implementadas na prática para resolver os problemas computacionalmente difíceis. No entanto, um estudo teórico sistemático deles permaneceu uma incógnita até recentemente. Uma das razões para isto é que, se uma entrada para um problema NP-difícil pode ser processada em tempo polinomial para um equivalente de um tamanho muito menor (constante), em geral, em seguida, o algoritmo de pré-processamento pode ser combinado para, na verdade, resolver o problema em tempo polinomial provando que $P = NP$, o que é considerado improvável. No entanto, a situação em matéria de estudo sistemático mudou drasticamente com o advento da complexidade parametrizada. A complexidade parametrizada fornece um quadro natural para analisar algoritmos de pré-processamento. Em um problema com parâmetros, cada instância x vem com um inteiro positivo, ou parâmetro, k. O problema é dito admitir um núcleo (kernel) se, em tempo polinomial, puder reduzir o tamanho da instância x a uma função de k, preservando ao mesmo tempo a resposta.

A noção central na complexidade parametrizada é a tratabilidade por parâmetro fixo (FPT), que é a noção de solucionabilidade em tempo $f(k).p(|x|)$ para qualquer instância (x,k), onde f é uma função arbitrária do parâmetro k e p é uma função polinomial no tamanho de entrada $|x|$. Resultados teóricos mostram que um problema parametrizado Π é tratável por parâmetro fixo se e somente se existe uma função computável $g(k)$ de tal forma que Π admite um núcleo de tamanho $g(k)$, i.e., algoritmos de pré-processamento que reduzem a entrada em uma instância de tamanho $g(k)$. No entanto, os núcleos obtidos por estes resultados teóricos são geralmente de tamanhos exponenciais (ou até pior) em relação ao parâmetro, enquanto reduções de dados específicos do problema muitas vezes alcançam núcleos quadráticos ou mesmo de tamanho linear. Assim, uma pergunta

natural para qualquer problema concreto em FPT é se tal problema admite redução a um núcleo (kernelization) de tempo polinomial para um núcleo do problema que, no pior caso é delimitada por uma função polinomial do parâmetro. Apesar de várias tentativas, há alguns problemas tratáveis por parâmetro fixo que somente se conhecem núcleos de tamanho exponenciais. Uma explicação foi fornecida em artigo por Bodlaender et al. (2009) [4], onde foi mostrado que a menos que $coNP \subseteq NP/poli$, há problemas tratáveis por parâmetro fixo que não pode ter um núcleo de tamanho polinomial. Isso desencadeou novos trabalhos mostrando limites para algoritmos de pré-processamento (algoritmos de redução a um núcleo ou algoritmos de kernelização).

5.8.1. Núcleos Polinomiais

Algoritmos de redução a um núcleo (Kernelização) são algoritmos de pré-processamento que satisfazem algumas condições adicionais. Observe que kernelização é uma redução de tempo polinomial de um problema para ele mesmo com a propriedade adicional que o tamanho da imagem é delimitado em termos do parâmetro do argumento. De fato, tratabilidade parametrizada e kernelização possuem noções idênticas. O teorema central para o entendimento entre kernelização e tratabilidade parametrizada pode ser encontrado em [13, 14, 20].

Teorema 14 *Para todo problema parametrizado* Π*, as seguintes afirmações são equivalente:*

1. $\Pi \in$ *FPT.*

2. Π *é decidível e* Π *admite um algoritmo de kernelização.*

Tendo que FPT é a classe de problemas redutíveis a um núcleo, temos agora uma outra maneira de analisar mais profundamente esta classe. Por exemplo, utilizando o tamanho dos núcleos obtidos como base para classificação.

Um procedimento de redução a um núcleo é usualmente descrito como uma coleção de regras de redução, que são desenvolvidas para transformar as instâncias e preservar algumas propriedades. Cada regra é geralmente aplicada recursivamente, onde cada aplicação encolhe a instância de alguma maneira, onde espera-se ser possível provar que se as regras forem aplicadas até não ser possível aplicá-las então a instância resultante deve ser um núcleo.

Naturalmente, gostaríamos que o núcleo produzido fosse o menor possível a ser obtido por um algoritmo de kernelização. Isto motiva a noção de núcleos polinomiais.

Núcleo Polinomial. Um núcleo polinomial é um núcleo cujo tamanho é polinomial em relação ao parâmetro original.

Conforme vimos anteriormente, o problema COBERTURA POR VÉRTICES parametrizado pelo tamanho da cobertura buscada é um exemplo de problema que admite núcleo polinomial.

5.8.2. Nenhum núcleo polinomial

Agora, estamos prontos para examinar as circunstâncias sob as quais não esperamos um núcleo polinomial para o problema parametrizado.

Esta seção descreve a caracterização de problemas que provavelmente não admitem núcleo de tamanho polinomial, utilizando como ferramenta uma certa propriedade. Esta propriedade tornou-se uma ferramenta muito útil, pois para exibir que um problema provavelmente não admite núcleo polinomial, basta demonstrar o problema em questão satisfaz esta propriedade. O resultados destas seção baseiam-se nos trabalhos seminais de Bodlaender et al. [4] e Fortnow e Santhanam [15].

Enquanto resultados positivos sobre a existência de núcleos polinomiais têm aparecido regularmente nas duas últimas décadas, os primeiros resultados estabelecendo a inviabilidade de núcleo polinomiais para problemas específicos têm aparecido somente recentemente. Em particular, Bodlaender et al. [4] e Fortnow e Santhanam [15] desenvolveram um framework baseado na noção de *composicionalidade*, para mostrar que um problema não admite um núcleo polinomial a menos que $NP \subseteq coNP/poly$, implicando um colapso na hierarquia polinomial até o terceiro nível ($PH = \Sigma_p^3$), que é considerado improvável.

Um *algoritmo de Ou-destilação* para uma dado problema é concebido para atuar como um "operador booleano Ou de instâncias de um problema", ele recebe como entrada uma sequência de instâncias do problema, e produz uma instância-SIM se e somente se pelo menos uma das instâncias da sequência dada também é uma instância-SIM. Embora seja permitido executar o algoritmo em tempo polinomial em relação ao tamanho total da sequência de entrada, a saída do algoritmo deve ser uma instância cujo tamanho é polinomialmente limitado pelo tamanho da maior instância da sequência de entrada. Formalmente, temos o seguinte:

Definição 14 *(Ou-destilação [4]) Seja Π um problema NP-completo. Uma Ou-destilação de Π é um algoritmo de tempo polinomial D que recebe como entrada uma série de m instâncias de Π, e retorna uma outra instância de Π, tal que*

- *Se D possui como entrada m instâncias, cada uma de tamanho no máximo n, então D utiliza tempo polinomial em m e n, e sua saída é delimitada por uma função que é polinomial em n.*
- *Se D têm como entrada instâncias $x_1, \ldots, x_m$, então $D(x_1, \ldots, x_m) \in \Pi$ se e somente se $\exists_{1 \leq i \leq m} x_i \in \Pi$.*

Teorema 15 *([4]) Se um problema NP-completo possui um algoritmo de Ou-destilação então $NP \subseteq coNP/poly$.*

Combinado com o teorema de Yap [22] ($NP \subseteq coNP/poly \Rightarrow PH \subseteq \Sigma_3^p$), temos que uma Ou-destilação para um problema NP-completo também implica em um colapso da hierarquia polinomial até o terceiro nível.

Definição 15 *(Ou-composição [4]) Seja $\Pi \subseteq \Sigma^* \times N$ um problema parametrizado. Uma Ou-composição de Π é um algoritmo de tempo polinomial D que recebe como entrada*

uma sequencia $((x_1,k),(x_2,k),\ldots(x_m,k))$*, com cada* $(x_i,k) \in \Sigma^* \times N$*, e retorna um par* (x',k')*, tal que*

- *o algoritmo utiliza tempo polinomial em* $\sum_{1\leq i\leq m}|x_i|+k$*;*
- k' *é delimitado por um polinomio em k;*
- $(x',k') \in \Pi$ *se e somente se* $\exists_{1\leq i\leq m}(x_i,k) \in \Pi$*.*

Um problema parametrizado é ou-composicional se ele admite um algoritmo de ou-composição. Se a versão parametrizada de um problema NP-completo admite tanto uma ou-composição quanto um núcleo polinomial, então o problema NP-completo possui uma ou-destilação. Isto implica que se um problema parametrizado possui um algoritmo de ou-composição, então ele não pode admitir núcleo polinomial, a menos que $NP \subseteq coNP/poly$, devido ao Theorem 15.

Teorema 16 *([4]) Seja* $\Pi(k)$ *um problema parametrizado ou-composicional tal que* Π *é NP-completo. Se* Π *possui núcleo polinomial, então* Π *também possui um algoritmo de ou-distilação.*

Algoritmos que compõem multiplas instâncias de um problema em uma única instância têm sido desenvolvidas para vários problemas [4, 19, 5, 6]. Exemplos de técnicas de ou-composição podem ser encontradas em [19]: composição por união disjunta, composição utilizando IDs, composição com cores e IDs, e composição como programação dinâmica.

Exemplo de Ou-composição.

Considere o seguinte problema:

Caminho(k)
Instância: Um grafo $G = (V,E)$.
Parâmetro: Um inteiro não negativo k.
Questão: G possui um caminho de comprimento k?

Este problema é conhecido pertencer a FPT com tempo de execução $2^{O(k)}n^{O(1)}$ via uma técnica conhecida como "color coding" [1].

A seguir apresentamos uma ou-composição para o problema CAMINHO(k). Considere as seguintes instâncias de entrada:

$$(G_1,k),\ldots,(G_t,k).$$

Uma algoritmo de ou-composição é trivial, ele simplesmente fornece a união disjunta G' de todos os grafos de entrada como saída. Note que esta operação consome tempo linear, e não altera o valor do parâmetro. Claramente, G' possui um caminho de comprimento no máximo k se e somente se existe i, $1 \leq i \leq t$, tal que G_i possui um caminho de comprimento no máximo k. Logo CAMINHO(k) não admite núcleo polinomial a menos que $NP \subseteq coNP/poly$.

A Ou-composição foi a primeira técnica a ser desenvolvida para demonstrar que alguns problemas em FPT não admitem núcleo polinomial, a menos que $NP \subseteq coNP/poly$. Após o seu surgimento, outras técnicas também foram desenvolvidas tais como PPT-reduções e composição-cruzada (cross-composition). [12, 9]

5.9. Complexidade Parametrizada × Algoritmos de Aproximação

Conforme [2], um problema NP de otimização (em NPO) é formalmente definido como uma 4-tupla $(I, sol, custo, meta)$, onde

- I é um conjunto de instâncias;
- para uma instância $x \in I$, $sol(x)$ é o conjunto de soluções viáveis de x. O tamanho de cada $y \in sol(x)$ é polinomialmente delimitado em $|x|$, e dado x e y, pode ser decidido em tempo polinomial se $y \in sol(x)$;
- dado uma instância x e uma solução viável y, $custo(x,y)$ é uma função inteira positiva computável em tempo polinomial;
- meta é ou *min* ou *max*;

O objetivo é encontrar uma solução viável y que alcança o melhor valor objetivo, isto é,

$$cost(x,y) = goal\{cost(x,y') : y' \in sol(x)\}.$$

O custo da solução ótima para uma instância x é denotada por $opt(x)$. Se y é uma solução para a instância x, então a razão de desempenho de y, $R(x,y)$, é definida como

$$cost(x,y)/opt(x) se\ goal\ =\ min,$$
$$opt(x)/cost(x,y) se\ goal\ =\ max.$$

Assim, $R(x,y)$ é sempre pelo menos 1; quanto mais próximo de 1, mais próximo a solução está do ótimo. Para um número real $c > 1$, dizemos que uma algoritmo é c-aproximado, se sempre produz uma solução com razão de desempenho no máximo c.

Dizemos que um problema X admite um esquema de aproximação de tempo polinomial (PTAS) se para todo $\varepsilon > 0$, existe um $(1+\varepsilon)$-algoritmo polinomial para X. Mais precisamente, existe um algoritmo que produz uma solução arbitrariamente boa. Isto é, um algoritmo A tal que dado uma instância x de X e um $\varepsilon > 0$, produz uma solução $(1+\varepsilon)$-aproximada em tempo $|x|^{f(1/\varepsilon)}$ para alguma função f. Claramente, tal algoritmo é executado em tempo polinomial para todo valor fixo de ε.

Se ε é pequeno então o expoente do polinômio $|x|f(1/\varepsilon)$ pode ser muito grande. Um esquema de aproximação de tempo polinomial eficiente (EPTAS) é um PTAS com tempo de execução da forma $f(1/\varepsilon).|x|^{O(1)}$.

Como sabemos, muitas vezes é mais fácil trabalhar com problemas de decisão ao invés de problemas de otimização, além disso a teoria da complexidade é baseada em problemas de decisão. No entanto, há uma forma padrão na qual problemas de otimização podem ser transformados em problemas de decisão equivalente. Se X é um problema de minimização (ou maximização), então definimos um problema de decisão como: uma instância x de X, um inteiro k e a seguinte questão "$opt(x) \leq k$?" (ou "$opt(x) \geq k$?").

Se X é um problema de otimização, a parametrização padrão de X é o problema parametrizado por k. Se a parametrização padrão é tratável por parâmetro fixo, então isto significa que temos um algoritmo eficiente para determinar o ótimo para instâncias onde o ótimo é pequeno.

Teorema 17 *(Bazgan [3]; Cesati e Trevisan [7]) Se um problema de otimização em NPO, X, admite EPTAS, então sua parametrização padrão é tratável por parâmetro fixo.*

Prova. Assuma que temos um algoritmo que produz uma solução $(1+\varepsilon)$-aproximada em tempo $f(1/\varepsilon).|x|^{O(1)}$. Dado uma instância (x,k) da versão parametrizada padrão de X, assumimos $\varepsilon = 1/(k+1)$, e utilizamos o EPTAS para encontrar uma solução $(1+\varepsilon)$-aproximada em tempo $f(1/\varepsilon).|x|^{O(1)} = f(k+1).|x|^{O(1)}$. Sem perda de generalidade, assuma que X é um problema de minimização. Se o ótimo é no máximo k, então o custo de uma solução aproximada é no máximo $(1+\varepsilon)k = k+(k/k+1) < k+1$. Como o custo é um inteiro, o custo da solução aproximada é no máximo k. Se o ótimo é maior do que k, então o custo da solução aproximada é também maior que k. Portanto, checando se o custo da solução aproximada é no máximo k, podemos decidir se o ótimo é no máximo k.

Podemos utilizar a contrapositiva para mostrar que é improvável que um problema particular admite um EPTAS.

Corolário 18 *Se a parametrização padrão de um problema de otimização é W[1]-difícil, então o problema de otimização não admite um EPTAS (a menos que FPT = W[1]).*

Observe que o inverso do Teorema 17 não é verdade. Por exemplo, determinar a cobertura mínima por vértices pertence a FPT, mas por outro lado é APX-difícil, e portanto não admite um PTAS. Sendo assim, Teorema 17 possui aplicabilidade limitada.

5.10. Considerações finais e leitura adicional

Esperamos, com este texto, poder ter fornecido uma visão inicial a área de complexidade parametrizada e, com isso, despertar maior interesse por este tópico. Se por um lado esta é uma área de pesquisa ainda em crescimento, por outro lado já pode ser considerada uma área consolidada, com resultados importantes e algumas técnicas bem compreendidas pela comunidade.

Embora os primeiros resultados na área de complexidade parametrizada datem de cerca de 30 anos atrás, a consolidação da área se deu após a publicação do primeiro livro dedicado exclusivamente a este tópico, escrito por Downey e Fellows. Embora já desatualizado, uma vez que retrata o estado da área até 1999, o texto ainda serve de texto

introdutório para um primeiro contato com área [13]. Mais recentemente, em 2013, uma nova versão do livro foi lançada [12], e fornece uma visão atualizada da área, embora diversos tópicos não sejam cobertos em profundidade, devido à abrangência do livro.

Dois outros livros, publicados em 2006, de Flum e Grohe [14] e de Niedermeier [20], fornecem possibilidades interessantes de aprofundamento na área. Em particular, o livro de Flum e Grohe é mais focado em apectos de complexidade e lógica, e pode ser mais adequados para os leitores com interesse nestas áreas.

Uma outra referência interessante é o livro de Cygan et al [9], em fase de elaboração. Ao contrário dos livros de Downey e Fellows, este livro abre mão de uma abrangência exageradamente grande, em troca de uma apresentação mais didática.

Acreditamos que estas fontes sejam referências adequadas para o próximo passo no aprofundamento do conhecimento daqueles que se interessarem pelo tema.

References

[1] Noga Alon, Raphael Yuster, Uri Zwick, Color-coding, J.ACM, v. 42(4), 844-856, 1995.

[2] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, M. Protasi, Complexity and Approximation, Springer-Verlag, Berlin, 1999.

[3] C. Bazgan. Schémas d'approximation et complexité paramétrée. Technical report, Université Paris Sud, 1995.

[4] Hans L. Boadlaender, Rodney G. Downey, Michael R. Fellows, Danny Hermelin, On problems without polynomial kernels, *Journal of Computer and System Sciences*, v. 75, p. 423-434, 2009.

[5] Hans L. Boadlaender, Stéphan Thomassé, Anders Yeo, Kernel bounds for disjoint cycles and disjoint paths, *Theoretical Computer Science*, v. 412, p. 4570-4578, 2011.

[6] Hans L. Bodlaender, Bart M.P. Jansen, Stefan Kratsch, Kernel bounds for path and cycle problems, *Theoretical Computer Science*, doi:10.1016/j.tcs.2012.09.006, 2012.

[7] M. Cesati, L. Trevisan. On the efficiency of polynomial time approximation schemes. Inform. Process. Lett., v. 64(4), p. 165-171, 1997.

[8] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. *Introduction to Algorithms*. MIT Press, Terceira edição, 2009.

[9] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk and Saket Saurabh. *Parameterized Algorithms*. A ser publicado, Springer, 2015+.

[10] Rodney G. Downey, Michael R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *Siam Journal on Computing*, **v. 24**, n. 4, p. 873-921, ago. 1995.

[11] Rodney G. Downey, Michael R. Fellows. Fixed-parameter tractability and completeness II: On completeness for W[1]. *Theoretical Computer Science*, **v. 141**, p. 109-131, abr. 1995.

[12] Rodney G. Downey, Michael R. Fellows. *Fundamentals of Parameterized Complexity*, Springer, 2013.

[13] Rodney G. Downey, Michael R. Fellows. *Parameterized complexity*, Monographs in Computer Science, Springer, 1999.

[14] Jörg Flum; Martin Grohe. *Parameterized complexity theory*, Springer, 2006.

[15] L. Fortnow, R. Santhanam, Infeasibility of instance compression and succinct PCPs for NP, *Journal of Computer and System Sciences*, v.77, p. 91-106, 2011.

[16] Michael R. Garey, David S. Johnson. *Computer and intractability. A Guide to the NP-Completeness*. Ney York, NY: WH Freeman and Company, 1979.

[17] Richard M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, 1972.

[18] H. W. Lenstra Jr. Integer programming with a fixed number of variables. *Mathematics of operations research* **8**(4), 538?548 (1983).

[19] Neeldhara Misra, Venkatesh Raman, Saket Saurabh, Lower Bounds on Kernelization, *Discrete Optimization*, v. 8, p. 110-128, 2011.

[20] Rolf Niedermeier. *Invitation to fixed-parameter algorithms*, Oxford Lecture Series in Mathematics and Its Applications, Oxford University Press, 2006.

[21] H. T. Wareham. Systematic parameterized complexity analysis in computational phonology, PhD thesis, University of Victoria, 1999.

[22] Chee-Keng Yap, Some Consequences of Non-Uniform Conditions on Uniform Classes, *Theoretical Computer Science*, v. 26, p. 287-300, 1983.