UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

NICOLLAS MOCELIN SDROIEVSKI

PROBLEMAS CANDIDATOS A NP-INTERMEDIÁRIOS E O PROBLEMA DE MINIMIZAÇÃO DE CIRCUITOS

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA

2016

NICOLLAS MOCELIN SDROIEVSKI

PROBLEMAS CANDIDATOS A NP-INTERMEDIÁRIOS E O PROBLEMA DE MINIMIZAÇÃO DE CIRCUITOS

Trabalho de Conclusão de Curso apresentado à disciplina de Trabalho de Conclusão de Curso 2 (TCC2) do Departamento Acadêmico de Informática da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do grau de "Bacharel em Sistemas de Informação"

Orientador: Prof. Murilo V. G. da Silva

CURITIBA

2016

RESUMO

SDROIEVSKI, Nicollas M.. PROBLEMAS CANDIDATOS A NP-INTERMEDIÁRIOS E O PROBLEMA DE MINIMIZAÇÃO DE CIRCUITOS. 70 f. Trabalho de Conclusão de Curso – Bacharelado em Sistemas de Informação, Universidade Tecnológica Federal do Paraná. Curitiba, 2016.

Neste trabalho é realizada fundamentação teórica do estado da arte da área de Complexidade Computacional, com foco para as classes definidas em torno de conceitos de probabilidade discreta. Mais especificamente são estudados quatro problemas candidatos a **NP**-intermediários, os problemas de minimização de circuitos, isomorfismo de grafos, resíduo quadrático e logaritmo discreto. Por último é exposto o poder de um oráculo para o poder de minimização de circuitos, e são mostrados explicitamente dois algoritmos aleatorizados polinomiais com oráculo para o problema de minimização de circuitos, cuja existência era conhecida apenas de forma indireta. O primeiro algoritmo resolve o problema do resíduo quadrático e o segundo o problema do logaritmo discreto.

Palavras-chave: Complexidade Computacional, **NP**-intermediário, Minimização de Circuitos, Oráculos, Algoritmos Aleatorizados Polinomiais, Isomorfismo de Grafos, Resíduo Quadrático, Logaritmo Discreto

ABSTRACT

SDROIEVSKI, Nicollas M.. NP-INTERMEDIATE CANDIDATE PROBLEMS AND THE MINIMUM CIRCUIT SIZE PROBLEM . 70 f. Trabalho de Conclusão de Curso – Bacharelado em Sistemas de Informação, Universidade Tecnológica Federal do Paraná. Curitiba, 2016.

In this work we study the state of the art of Computational Complexity, focusing on classes defined around discrete probability concepts. More specifically we study four problems that are **NP**-intermediate candidates, the minimum circuit size problem, graph isomorphism, quadratic residue and discrete logarithm. We also expose the power that an oracle for the minimum circuit size problem possesses, and show explicitly two randomized polynomial time algorithms with oracle access to the minimum circuit size problem, whose existence was only indirectly known. The first algorithm solves the quadratic residue problem and the second one solves the discrete logarithm problem.

Keywords: Computational Complexity, **NP**-intermediate, Minimum Circuit Size Problem, Oracles, Randomized Polynomial time Algorithms, Graph Isomorphism, Quadratic Residue, Discrete Logarithm

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Intuitivamente, pode-se perceber que existem problemas computacionais que são mais difíceis de resolver que outros, como é o caso de realizar a multiplicação de dois números versus a adição desses mesmos dois números, onde é percebida a maior dificuldade do primeiro tipo de operação.

A Complexidade Computacional é o ramo da Teoria da Computação que visa classificar problemas computacionais de acordo com sua dificuldade inerente, que pode ser expressa através do número de operações computacionais básicas realizadas pelo melhor algoritmo que decide o problema. Esse tipo de classificação é importante pois a eficiência de um algoritmo é em geral muito mais importante do que a tecnologia utilizada para executá-lo.

Um problema central de Complexidade Computacional é descobrir se as classes **P**, problemas que são fáceis[1] de resolver, e **NP**, problemas que são fáceis de verificar, definidas formalmente no capítulo 2, são equivalentes ou não, isto é, se $\mathbf{P} = \mathbf{NP}$ ou $\mathbf{P} \neq \mathbf{NP}$. Muitos pesquisadores acreditam que $\mathbf{P} \neq \mathbf{NP}$, porém até hoje não foi possível encontrar uma resposta concreta para esse problema.

Uma forma interessante de olhar para esse problema é a seguinte: Será que é tão fácil resolver um jogo de Sudoku qualquer do zero quanto verificar uma solução já preenchida? Ou então, será que é tão fácil provar um teorema matemático quanto verificar a validade de uma prova apresentada? As implicações de uma resposta positiva são incrivelmente utópicas (ARORA; BARAK, 2009), e mesmo assim o problema continua em aberto.

Devido a estagnação das tentativas de prova para a questão **P** versus **NP**, muitos pesquisadores começaram a estudar classes de complexidade diferentes, muitas vezes definidas através de modelos não clássicos de computação. Esse é o caso das classes de complexidade definidas através de circuitos Booleanos ($\mathbf{P_{/poly}}$), computação aleatória

[1] Levam tempo polinomial.

(**BPP**) e provas interativas (**IP**). E interessantemente muitas dessas classes possuem relações com as classes de complexidade mais tradicionais. De fato, grande parte dos estudos realizados atualmente visa relacionar as várias classes de complexidade existentes, e além disso provar a pertinência de problemas computacionais nessas classes. Um dos objetivos desse trabalho é realizar um levantamento do estado da arte da área de Complexidade Computacional.

Alguns problemas são candidatos a pertencer a uma classe conhecida como **NP**-intermediária, classe na qual estariam problemas em $\mathbf{NP} \setminus \mathbf{P}$, porém não **NP**-completos. Ladner (1975) mostrou que, caso $\mathbf{P} \neq \mathbf{NP}$, então existem problemas que se encaixam nessa classificação. Porém mostrar diretamente que um problema pertence à essa classe é extremamente difícil, pois implicaria que $\mathbf{P} \neq \mathbf{NP}$.

Nesse trabalho também estudamos a relação entre quatro problemas candidatos a **NP**-intermediários, os problemas de minimização de circuitos, isomorfismo de grafos, resíduo quadrático e logaritmo discreto. Em especial, Allender e Das (2014) mostraram, de maneira direta, como um oráculo para o problema de minimização de circuitos pode ser utilizado para obter um algoritmo aleatorizado polinomial para o problema de isomorfismo de grafos, e de maneira indireta algoritmos aleatorizados polinomiais para os problemas do resíduo quadrático e logaritmo discreto.

Este trabalho possui uma contribuição original: apresentamos explicitamente dois algoritmos polinomiais com oráculo para o problema de minimização de circuitos. O primeiro algoritmo resolve o problema do resíduo quadrático e o segundo o problema do logaritmo discreto. Como já comentado, o fato de que esses algoritmos devam existir, ainda que de maneira indireta, é conhecido. Os algoritmos apresentados neste trabalho são bastante simples e diretos e deixam bastante claro o papel que um oráculo para o problema de minimização de circuitos tem na resolução dos dois problemas.

1.1 JUSTIFICATIVA

O tema deste trabalho é de natureza teórica/científica, e este busca contribuir com a comunidade acadêmica através da produção de texto científico. Espera-se ainda que o estudo da relação entre os problemas candidatos a **NP**-intermediários citados possa contribuir para a resolução de questões mais complexas da área de Complexidade Computacional, como a questão **P** versus **NP**.

1.2 OBJETIVO GERAL

O objetivo geral do trabalho é realizar um estudo da área de Complexidade Computacional, seus fundamentos e estado da arte, permitindo então estudar mais profundamente a relação entre os problemas candidatos a **NP**-intermediários citados.

1.3 OBJETIVOS ESPECÍFICOS

Os objetivos específicos deste trabalho são os seguintes:

- Realizar fundamentação teórica da área de Complexidade Computacional, com o objetivo de permitir o entendimento da principal contribuição do trabalho.
- Estudar as relações conhecidas entre os problemas de minimização de circuitos, isomorfismo de grafos, resíduo quadrático e logaritmo discreto.
- Como principal contribuição: mostrar explicitamente como um oráculo para o problema de minimização de circuitos pode ser usado para resolver os problemas do resíduo quadrático e do logaritmo discreto.

1.4 ORGANIZAÇÃO DO TRABALHO

Este trabalho está organizado da seguinte maneira: no capítulo 2 são apresentados conceitos de complexidade computacional, desde definições básicas até modelos computacionais não uniformes. Esses conceitos são essenciais para entendimento do trabalho, pois através deles é definida a questão **P** versus **NP** e o que são problema candidatos a **NP**-intermediários.

No capítulo 3 é apresentada a técnica de diagonalização, suas características e limitações. A técnica de diagonalização é a técnica utilizada para provar que caso $\mathbf{P} \neq \mathbf{NP}$, então existem problemas **NP**-intermediários, o que torna essa técnica central para discussão desse trabalho.

No capítulo 4 são apresentados conceitos de Complexidade Computacional e aleatoriedade, importantes para os objetivos deste trabalho pois ajudam a definir classes de complexidade aleatórias, às quais os algoritmos apresentados pertencem. Além de conceitos importantes como protocolos de conhecimento zero e funções unidirecionais, que são utilizados na construção dos algoritmos.

No capítulo 5 são apresentados quatro problemas candidatos a **NP**-intermediários. No capítulo 6 mostramos porque um oráculo para o problema de minimização de circuitos é bastante poderoso e no capítulo 7 são apresentados os dois algoritmos aleatorizados polinomiais com oráculo para o problema de minimização de circuitos, que são a contribuição principal desse trabalho.

2 CONCEITOS DE COMPLEXIDADE COMPUTACIONAL

Neste capítulo são apresentados conceitos importantes da área de Complexidade Computacional, além de definições formais e teoremas que ou serão utilizados durante o decorrer do trabalho ou são importantes para compreender as ideias principais desse. Todas as definições, a menos que indicado o contrário, são baseadas em (ARORA; BARAK, 2009).

2.1 DEFINIÇÕES BÁSICAS E NOTAÇÃO

Medir a complexidade de um algoritmo envolve a medida do tempo de execução desse como uma função em relação ao tamanho da entrada. Dessa forma, um algoritmo executando com uma entrada de tamanho n que executa até $T(n)$ passos computacionais terá uma medida de complexidade $T(n)$, onde T é uma função nos números naturais. Porém, essa função $T(n)$ é muito dependente de detalhes de baixo nível da implementação do algoritmo, portanto é interessante usar notação assintótica para medir a complexidade de algoritmos.

Definição 2.1. (Notação Assintótica) Sejam f e g funções de $\mathbb{N}$ para $\mathbb{N}$, (1) dizemos que $f = O(g)$ se existe uma constante $c > 0$ tal que $f(n) \leq c \cdot g(n)$ para n suficientemente grande. (2) Dizemos que $f = \Omega(g)$ se $g = O(f)$, (3) que $f = \Theta(g)$ se $f = O(g)$ e $f = \Omega(g)$, (4) que $f = o(g)$ se para todo $\epsilon > 0$, $f(n) \leq \epsilon \cdot g(n)$ para n suficientemente grande e (5) que $f = \omega(g)$ se $g = o(f)$.

Para dar destaque ao tamanho da entrada de um algoritmo, algumas vezes será escrito $f(n) = O(g(n))$ ao invés de $f = O(g)$, notação similar pode ser utilizada para Ω, Θ, o e ω.

Uma notação importante para o trabalho é a de representação de objetos através de strings binárias, que será indicado através do símbolo $\llcorner O \lrcorner$ em volta do objeto O em questão, que pode ser um grafo, número inteiro, ou qualquer outro objeto representável por uma quantidade finita de símbolos. Além disso, será utilizada a notação

$\langle O_1, O_2, \ldots, O_n \rangle$ para representar a concatenação das strings em binário que representam os objetos $O_1, O_2, \ldots, O_n$.

Outra definição importante é a de grafos: um grafo G (WEST, 2000) consiste em um conjunto $V(G)$ (ou apenas V) de *vértices*, que normalmente consideramos como o conjunto $[n] = \{0, \ldots, n\}$ para algum $n \in \mathbb{N}$, e um conjunto de *arestas* $E(G)$ (ou apenas E), que consiste de pares não-ordenados de $V(G)$. Denominamos a aresta $\{u, v\}$ do grafo por $\overline{u\ v}$. Para $v \in V(G)$, os *vizinhos* de v são todos os vértices $u \in V(G)$ tal que $\overline{u\ v} \in E(G)$. Em um grafo *dirigido*, as arestas são pares ordenados de vértices, e para dar destaque algumas vezes denotamos a aresta $\langle u, v \rangle$ como $\overrightarrow{u\ v}$. Na Figura 1 pode ser observado um exemplo de grafo:

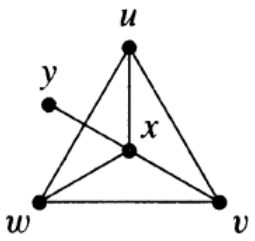

Figura 1: Exemplo de grafo.
Fonte: (WEST, 2000).

$$\begin{array}{c} \begin{array}{ccccc} u & v & w & x & y \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{array}{c} u \\ v \\ w \\ x \\ y \end{array} \end{array}$$

Figura 2: Matriz de adjacência do grafo à esquerda.
Fonte: Autoria própria.

Pode-se representar um grafo G de n vértices através de sua *matriz de adjacência*, que é uma matriz $n \times n$ A, tal que $A_{i,j}$ é igual a 1 se a aresta $\overrightarrow{i\ j}$ está presente em G e igual a 0 caso contrário. Pode-se pensar em um grafo não-direcionado como um grafo direcionado G que satisfaz a condição de que para todos $u, v \in V$, G contém a aresta $\overrightarrow{u\ v}$ se e somente se também contém a aresta $\overrightarrow{v\ u}$. Por esse motivo, pode-se representar um grafo não direcionado por uma matriz de adjacência simétrica ($A_{i,j} = A_{j,i}$ para todos $i, j \in [n]$). A Figura 2 mostra um exemplo de matriz de adjacência.

Podemos *permutar* os vértices de um grafo G, obtendo, potencialmente, um novo grafo G'. Denominamos por S_n o conjunto de todas as permutações em n vértices, e se $\pi \in S_n$, denotamos por $\pi(G)$ o resultado da aplicação da permutação π ao grafo G. Podemos ainda compor duas permutações π_1 e π_2, que denotamos $\pi_1 \circ \pi_2$.

Por último, ainda apresentamos algumas definições relacionadas a grupos. Grupos são abstrações que capturam propriedades de objetos matemáticos como números inteiros, matrizes, funções e outros. Formalmente um grupo é definido por um conjunto e

uma operação binária associativa para a qual todo elemento do grupo possui um inverso. Grupos finitos são aqueles que possuem um número finito de elementos, um exemplo é o conjunto S_n de permutações com a operação binária de composição. Subgrupos são subconjuntos de um grupo que mantém as propriedades acima.

Um grupo finito importante para esse trabalho é o grupo $\mathbb{Z}_n^*$, que consiste no conjunto $\{k; 1 \leq k \leq n-1 \text{ e } mdc(k,n) = 1\}$ com a operação binária de multiplicação módulo n. A condição de que o máximo divisor comum de k e n deve ser 1 é necessária pois caso contrário nem todo elemento do conjunto possuiria um inverso. No caso específico em que n é um número primo, o tamanho do grupo é $n-1$.

Quando podemos obter todos os elementos de um grupo G através da aplicação repetida da operação de G a um elemento $b \in G$, dizemos que b é um elemento gerador de G. No caso específico de $\mathbb{Z}_n^*$, quando n é primo o grupo sempre possui um elemento gerador.

2.2 MÁQUINAS DE TURING

Uma Máquina de Turing, abreviada como MT e definida pela primeira vez por Alan Turing (1936), é um modelo matemático para representar um algoritmo. Definir matematicamente um modelo de computação é essencial para compreender as ideias apresentadas nesse trabalho.

Informalmente, uma MT possui k fitas, infinitas para a direita, a primeira fita é denominada fita de *entrada* e a última fita é denominada fita de *saída*, as outras fitas são denominadas fitas de *trabalho*. Uma MT também possui um conjunto de símbolos que podem estar nessas fitas, um conjunto de estados possíveis e um conjunto de regras que definem como a máquina deve manipular os símbolos nas suas fitas e quando deve mudar de estado. Uma MT ainda possui k cabeças de leitura/gravação, que podem ler e modificar um símbolo por vez em cada uma de suas fitas. Mais formalmente, uma MT M é uma tupla (Γ, Q, δ), onde:

- Γ é o alfabeto das fitas de M, que será o conjunto $\{0, 1, \square, \triangleright\}$, onde $\square$ é o símbolo especial *branco* e $\triangleright$ é o símbolo que representa o início das fitas de M.
- Q é o conjunto de estados em que M pode estar, assumimos que M possui um estado inicial q_{start} e um estado de parada q_{halt}.
- $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{L, S, R\}^k$, onde $k \geq 2$, é a função de transição de M, que

define o seu comportamento em cada passo computacional.

Se M está no estado $q \in Q$, $(\sigma_1, \sigma_2, \ldots, \sigma_k)$ são os símbolos sendo lidos nas k fitas de M, e $\delta(q, (\sigma_1, \sigma_2, \ldots, \sigma_k)) = (q', (\sigma'_2, \ldots, \sigma'_k), z)$, onde $z \in \{L, S, R\}^k$, então no próximo passo, os símbolos σ de M em suas últimas $k-1$ fitas serão trocados pelo símbolo σ' correspondente, além disso M estará no estado q' e suas k cabeças vão se mover para a esquerda (L), direita (R) ou ficar na mesma posição (S), de acordo com z. Se M tenta mover alguma cabeça para a esquerda, estando na posição mais à esquerda da fita, a cabeça ficará na mesma posição.

Todas as fitas com exceção da fita de *entrada* são inicializadas com o símbolo $\triangleright$ na sua primeira posição, e o símbolo *branco* $\square$ em todas as outras posições. A fita de *entrada* contém, no início da computação, o símbolo $\triangleright$, uma *string* x (a entrada de M) e o símbolo *branco* $\square$ no resto de suas posições. Todas as cabeças começam na extremidade esquerda de suas fitas e a máquina começa no estado q_{start}. Cada passo computacional é realizado aplicando a função δ, e o estado q_{halt} possui a propriedade de que, uma vez que a máquina esteja nesse estado, a função de transição δ não mais permite que a máquina modifique suas fitas ou mude de estado, efetivamente *parando* sua execução.

Seja $\{0,1\}^*$ o conjunto de todas as strings binárias. Dizemos que uma MT M computa uma função $f : \{0,1\}^* \to \{0,1\}^*$, se para todo $x \in \{0,1\}^*$, ao inicializar M com x em sua fita de entrada, M para com o resultado $f(x)$ escrito em sua fita de saída. A saída de uma MT M pode ser ainda denotada por $\mathrm{M}(x)$ e dessa forma dizemos que M computa a função f se $\mathrm{M}(x) = f(x)$ para todo $x \in \{0,1\}^*$. Além disso, sendo T uma função nos naturais, dizemos que M computa f em tempo $T(n)$ se M executa no máximo $T(|x|)$ passos para qualquer entrada $x \in \{0,1\}^*$.

Limitamos o tempo de execução de MTs para funções *tempo-construtíveis* no caso de complexidade de tempo e *espaço-construtíveis* no caso de complexidade de espaço, pois permitir funções não-construtíveis em ambos os casos pode levar a resultados anômalos (ARORA; BARAK, 2009):

Definição 2.2. (Funções tempo-construtíveis) Uma função $T : \mathbb{N} \to \mathbb{N}$ é dita tempo-construtível se $T(n) \geq n$ e se existe uma MT M que computa a função $x \to \llcorner T(|x|) \lrcorner$ em tempo $T(n)$. Exemplos de funções tempo-construtíveis são n, $n \log n$, n^2, 2^n e quase todas as funções encontradas neste trabalho serão tempo-construtíveis. A restrição $T(n) \geq n$ é para que o algoritmo tenha tempo de ler sua entrada.

Definição 2.3. (Funções espaço-construtíveis) Uma função $S : \mathbb{N} \to \mathbb{N}$ é dita espaço-

construtível se $S(n) > \log n$ se existe uma MT M que computa a função $x \to \llcorner S(|x|) \lrcorner$ em espaço $S(n)$. Percebe-se, nesse caso, que não é necessário que $S(n) \geq n$, porém existe a restrição de que $S(n) \geq \log n$ para que uma MT possa pelo menos "lembrar"o índice da posição da fita de entrada que está lendo.

2.2.1 PROBLEMAS DE DECISÃO E LINGUAGENS

Uma Linguagem ou Problema de Decisão (HOPCROFT et al., 2001) é um conjunto $L \subseteq \{0,1\}^*$. Uma linguagem, apesar de ser sempre sobre o alfabeto $\{0,1\}$, pode representar outros tipos de objeto, como números inteiros ou grafos, através da sua codificação em binário. Por exemplo: $L = \{\llcorner G \lrcorner; G \text{ é um grafo conexo}\}$, onde $\llcorner G \lrcorner$ é a codificação binária do grafo G, é uma linguagem válida.

Nossa atenção estará voltada para máquinas de Turing que computam funções booleanas do tipo $f : \{0,1\}^* \to \{0,1\}$, isso é, dada uma MT M e entrada x, $M(x) = 1$ ou $M(x) = 0$ para qualquer $x \in \{0,1\}^*$. Dessa forma, dizemos que uma MT M *decide* uma linguagem $L \subseteq \{0,1\}^*$ se M computa a função booleana $f_L : \{0,1\}^* \to \{0,1\}$, onde $f_L(x) = 1 \leftrightarrow x \in L$.

2.2.2 VARIAÇÕES DE MÁQUINAS DE TURING

Algumas das escolhas da definição de MT da subseção anterior podem parecer arbitrárias e limitantes, como é o caso de restringir o alfabeto da MT ao alfabeto $\{0, 1, \square, \triangleright\}$, ou então muito poderosas, como é o caso de permitir uma quantidade arbitrária (finita) de fitas para a MT.

A questão é que mudanças nessas características de uma MT não irão alterar as classes de complexidade que serão definidas mais adiante, pois em geral esse tipo de mudança leva a um aumento no máximo polinomial do tempo de execução da MT (ARORA; BARAK, 2009). Exemplos desse tipo de mudança e o *overhead* de tempo de computação são tratados nas afirmações 2.1 e 2.2.

Afirmação 2.1. *(ARORA; BARAK, 2009) Para toda função $f : \{0,1\}^* \to \{0,1\}$ e função tempo-construtível $T : \mathbb{N} \to \mathbb{N}$, se f é computável em tempo $T(n)$ por uma máquina M que possui alfabeto Γ, então f é computável em tempo $4 \log |\Gamma| T(n)$ por uma MT M' que possui o alfabeto $\{0, 1, \square, \triangleright\}$.*

Afirmação 2.2. *(ARORA; BARAK, 2009) Definimos uma MT com uma fita como sendo uma MT que possui apenas uma fita de leitura/escrita que funciona como fita de entrada,*

trabalho e saída. Para toda função $f : \{0,1\}^* \to \{0,1\}$ *e função tempo-construtível* $T : \mathbb{N} \to \mathbb{N}$*, se* f *é computável em tempo* $T(n)$ *por uma máquina* M *que possui* k *fitas, então* f *é computável em tempo* $5kT(n)^2$ *por uma MT* M' *com uma fita.*

Além das afirmações acima, ainda é possível simular MTs que possuem fita infinita para os dois lados com *overhead* constante, máquinas com acesso indexado à suas fitas (no estilo RAM) e fitas tridimensionais com *overhead* polinomial. Também é possível garantir que toda MT M pode ser simulada por uma MT M' que é *alienada* (tradução livre de *oblivious*), no sentido de que o movimento das cabeças de M' depende somente do tamanho da entrada, e não da entrada especificamente (ARORA; BARAK, 2009).

Como um adendo, vale a pena citar a **Tese de Church-Turing**, que indica que MTs são capazes de simular a computação de qualquer meio físico (computadores, neurônios, o processo de *self-assembly* de moléculas de DNA), o que implica que o conjunto de funções *computáveis* não muda quando assumimos que o nosso modelo de computação é uma MT. Há ainda a **Tese de Church-Turing estendida**, que indica que essa simulação possui *overhead* no máximo polinomial. E até o momento apenas o modelo de computador quântico aparenta não ser eficientemente simulável em uma MT, porém não sabemos se realmente é possível construir um computador quântico.

2.2.3 MÁQUINAS DE TURING COMO STRINGS E MÁQUINA DE TURING UNIVERSAL

É fácil perceber que uma máquina de Turing pode ser representada por uma string, basta escrever sua descrição em um papel e então codificar essa descrição como sequência de zeros e uns. Mas o que é mais interessante é que essa string pode ser utilizada como a entrada de outra MT (ou até da mesma), e esse conceito é muito usado em várias provas de teoremas de Complexidade Computacional.

Ao definir um esquema de codificação para MTs, é interessante garantir duas propriedades: a primeira é a de que toda string $x \in \{0,1\}^*$ representa alguma MT (isso é fácil quando mapeamos strings que não codificam MTs para uma MT canônica qualquer, como a MT que para e rejeita qualquer string) e a segunda é que toda MT pode ser representada por infinitas strings (basta definir que a representação de qualquer MT pode terminar com um número arbitrário de caracteres 1, sem mudar seu comportamento). Em especial, denotaremos por $\llcorner M \lrcorner$ a codificação em binário da MT M, e se α é uma string, denotaremos por M_α a MT que a string α representa.

Alan Turing (1936) foi o primeiro a mostrar que era possível construir uma Máquina de Turing Universal, que de forma análoga a um computador moderno, consegue simular o comportamento de qualquer outra MT. Formalmente, uma MT Universal é uma MT que, ao receber como entrada uma string que possui a descrição de uma MT M e uma entrada x, simula o comportamento da máquina M ao rodar com x. É importante notar que é possível construir uma MT universal que simula qualquer outra MT M com *overhead* logarítmico, ou seja, executando uma quantidade de passos que supera a MT M por um fator logarítmico. O seguinte teorema formaliza essa noção:

Teorema 2.1. *(Máquina de Turing Universal Eficiente) (HENNIE; STEARNS, 1966) Existe uma MT U tal que para todo $x, \alpha \in \{0,1\}^*$, $U(\alpha, x) = M_\alpha(x)$. Além disso, se M_α para com a entrada x em até T passos, então $U(\alpha, x)$ para em até $cT \log T$ passos, onde c é uma constante que independe de x, e depende apenas da quantidade de estados, símbolos e fitas de M_α.*

A existência de uma MT Universal eficiente é muito importante para demonstrar vários resultados da área de Complexidade Computacional.

2.2.4 MÁQUINAS DE TURING COM ORÁCULOS

Máquinas de Turing com Oráculos são extensões de MTs que possuem acesso à um oráculo para determinada linguagem O. O oráculo permite à máquina decidir, em apenas um passo computacional, se determinada string pertence ou não à linguagem O. Como a linguagem O pode até mesmo ser indecidível, oráculos não possuem correspondência no mundo físico, porém são conceitos importantes no estudo de Complexidade Computacional. Mais formalmente:

Definição 2.4. (Máquina de Turing com Oráculo) Uma Máquina de Turing com Oráculo é uma MT M que possui uma fita especial de escrita e leitura que é chamada de *fita oráculo* de M e três estados q_{query}, q_{yes} e q_{no}. Para executar M, além da linguagem de entrada, é definida uma linguagem $O \subseteq \{0,1\}^*$ que é usada como *oráculo* de M. Quando M entra no estado q_{query}, a máquina se move para o estado q_{yes} se $q \in O$ e para o estado q_{no} se $q \notin O$, onde q é o conteúdo da fita oráculo de M. É importante notar que independente da escolha de O, uma consulta de pertinência a O conta somente como um passo computacional. Sejam M uma MT com Oráculo, $O \subseteq \{0,1\}^*$ uma linguagem e $x \in \{0,1\}^*$, então denotamos a saída de M com a entrada x e com oráculo O por $M^O(x)$.

Para toda linguagem $O \subseteq \{0,1\}^*$, podemos definir classes de complexidade com

oráculo para O. Por exemplo, $\mathbf{P}^O$ é o conjunto de todas as linguagens decidíveis em tempo polinomial por máquinas de Turing determinísticas com oráculo O.

2.3 A CLASSE P

A classe **P**, primeiramente definida com o nome $\mathscr{L}$ por Cobham (1964) engloba todos os problemas decidíveis em tempo polinomial, ou seja, os problemas para os quais existem MTs que executam no máximo um número polinomial de passos no tamanho da entrada. Antes de poder definir formalmente a classe **P**, é importante definir a classe **DTIME**:

Definição 2.5. (A classe **DTIME**) Seja $T : \mathbb{N} \to \mathbb{N}$ uma função qualquer, dizemos que uma linguagem L está em **DTIME**$(T(n))$ se e somente se existe uma MT de tempo $c \cdot T(n)$, para uma constante $c > 0$, que decide L.

Agora é possível definir formalmente a classe **P**:

Definição 2.6. (A classe **P**) **P**$= \bigcup_{c \geq 1}$**DTIME**(n^c).

O objetivo da classe **P** é capturar os problemas "fáceis", ou aqueles que são factíveis de serem computados. Por esse motivo, pode parecer estranho incluir nessa classe, por exemplo, a classe **DTIME**(n^{100}), que é proibitivamente lenta até para valores não tão grandes de n. Porém, normalmente quando é encontrado um algoritmo polinomial para algum problema, a complexidade de tempo desse fica entre n e n^5, e inclusive já aconteceu de o primeiro algoritmo encontrado ser n^{20}, porém posteriormente esse algoritmo foi melhorado para por exemplo, n^5 (ARORA; BARAK, 2009).

Outro argumento interessante é o de que como a maior parte dos pesquisadores da área de Complexidade Computacional acreditam que **P** é diferente da classe **NP**, que será definida a seguir, incluir classes como **DTIME**(n^{100}) só deixa o resultado, quando eventualmente provado, mais forte.

2.4 AS CLASSES NP E coNP

Definimos a classe **P**, que engloba os problemas de decisão para os quais existem algoritmos polinomiais, ou então os problemas fáceis de resolver. Existem, porém, problemas que se encaixam com mais facilidade na classe dos problemas verificáveis em tempo

polinomial (ou fáceis de verificar), que é a classe **NP**, cujas ideias básicas foram estudadas pela primeira vez por Cook (1971) e independentemente por Levin (1973), apesar de muitos pesquisadores já terem noção dos conceitos envolvidos desde 1950 (ARORA; BARAK, 2009).

Definição 2.7. (A classe **NP**) Uma linguagem $L \subseteq \{0,1\}^*$ está em **NP** se existe um polinômio $p : \mathbb{N} \to \mathbb{N}$ e uma MT M de tempo polinomial (chamada de *verificador* de L) tal que para todo $x \in \{0,1\}^*$:

$$x \in L \leftrightarrow \exists u \in \{0,1\}^{p(|x|)}; M(x,u) = 1.$$

Se $x \in L$ e $u \in \{0,1\}^{p(|x|)}$ satisfazem $M(x,u) = 1$, então chamamos u de *certificado* para a string x (em relação à linguagem L e à máquina M).

Percebe-se que os problemas em **NP** não só são verificáveis em tempo polinomial, como quando possuem uma solução, essa também possui tamanho polinomial (essa solução é o *certificado* de uma instância do problema).

Outra maneira de definir a classe **NP**, cuja sigla significa, em inglês, **P**olinomial **N**ão-determinística, depende da definição de uma Máquina de Turing Não-determinística, ou MTND. A principal diferença entre uma MT e uma MTND é que a segunda possui um estado especial q_{accept} e duas funções de transição δ_0 e δ_1. Quando uma MTND M computa uma função, ela realiza escolhas arbitrárias de qual função de transição aplicar. Para toda entrada x, dizemos que $\mathrm{M}(x) = 1$ se *existe* uma sequência de escolhas (que chamamos de escolhas não-determinísticas de M) que faça M alcançar o estado q_{accept} com a entrada x. Caso contrário, se *nenhuma* sequência de escolhas não-determinísticas de M rodando com a entrada x alcança o estado q_{accept}, então $\mathrm{M}(x) = 0$. Dizemos que M roda em tempo $T(n)$ se para toda entrada $x \in \{0,1\}^*$ e qualquer sequência de escolhas não-determinísticas, M ou alcança o estado q_{halt} ou o estado q_{accept} em até $T(|x|)$ passos.

Tendo a definição de uma MTND, é possível então definir a classe **NTIME** e dar uma definição alternativa para **NP**, que pode ser útil em algumas provas:

Definição 2.8. (A classe **NTIME**) Para toda função $T : \mathbb{N} \to \mathbb{N}$ e $L \subseteq \{0,1\}^*$, dizemos que $L \in \mathbf{NTIME}(T(n))$ se existe uma constante $c > 0$ e uma MTND M de tempo $c \cdot T(n)$ tal que para todo $x \in \{0,1\}^*$, $x \in L \leftrightarrow M(x) = 1$.

Definição 2.9. (A classe **NP**, definição alternativa) $\mathbf{NP} = \bigcup_{c \in \mathbb{N}} \mathbf{NTIME}(n^c)$

Tendo definido a classe **NP**, é interessante explorar sua relação com a classe **P**. Até o momento, a única relação conhecida é que $\mathbf{P} \subseteq \mathbf{NP}$, pois qualquer problema de **P**

está trivialmente em **NP**, fixando um certificado vazio ou então levando em consideração que uma MT é um caso especial de uma MTND em que as duas funções de transição δ_0 e δ_1 são iguais.

Existem, porém, problemas em **NP** para os quais não é conhecido nenhum algoritmo polinomial, e até o momento os melhores algoritmos conhecidos rodam em tempo exponencial. Interessantemente, descobriu-se que muitos desses problemas são pelo menos tão difíceis quanto qualquer outro problema em **NP** (KARP, 1972). Um problema desse tipo é chamado de **NP**-difícil, e se esse problema também pertence a **NP**, é chamado de **NP**-completo (COOK, 1971). Para poder definir formalmente quando um problema é tão difícil quanto outro, é importante definir a noção de *redução polinomial*:

Definição 2.10. (Redução polinomial) Uma linguagem $L \subseteq \{0,1\}^*$ é Karp-redutível em tempo polinomial para outra linguagem $L' \subseteq \{0,1\}^*$ (ou então redutível em tempo polinomial), denotado por $L \leq_p L'$, se existe uma função computável em tempo polinomial $f : \{0,1\}^* \to \{0,1\}^*$ tal que para todo $x \in \{0,1\}^*$, $x \in L \leftrightarrow f(x) \in L'$.

Podemos então definir quando um problema é **NP**-difícil ou **NP**-completo:

Definição 2.11. Uma linguagem $L' \subseteq \{0,1\}^*$ é **NP**-difícil se $L \leq_p L'$ para toda linguagem $L \in \mathbf{NP}$. Se L' é **NP**-difícil e $L' \in \mathbf{NP}$, então L' é **NP**-completa.

Além de definirem a classe **NP** pela primeira vez, ambos Cook (1971) e Levin (1973) mostraram que um problema combinatório natural, que aparentemente não tem relação com Máquinas de Turing ou Linguagens, é **NP**-completo. Esse problema é o problema da Satisfazibilidade Booleana, ou SAT, que é o problema de decidir se uma fórmula booleana dada em Forma Normal Conjuntiva pode ser satisfeita por alguma valoração de suas variáveis. Notavelmente, o problema 3-SAT, cujas instâncias são fórmulas booleanas com até três literais por cláusula, também é **NP**-completo, apesar de parecer mais limitado:

Teorema 2.2. *(Teorema Cook-Levin) (COOK, 1971; LEVIN, 1973) Os problemas SAT e 3-SAT são **NP**-completos.*

A prova do teorema Cook-Levin envolve o fato importante de que a computação é *local* (ARORA; BARAK, 2009). Isso é, a cada passo computacional, uma MT modifica apenas alguns bits de suas fitas. Mais adiante será apresentado um argumento da importância desse fato para buscar uma solução para o problema **P** versus **NP**.

Um ano após o artigo de Cook ser publicado, Karp (1972) mostrou que vários outros problemas combinatoriais, como conjunto independente e coloração em grafos, o problema da mochila e cobertura de conjuntos são **NP**-completos. Esse fato fez com que o problema **P** versus **NP** ganhasse muita importância e a atenção de muitos pesquisadores, pois resolver essa questão traria consequências para vários problemas com aplicações importantes.

Interessantemente, Karp cita em seu artigo que não conseguiu mostrar que o problema de isomorfismo de grafos é **NP**-completo, e muitas outras tentativas de provar que o problema é **NP**-completo foram frustradas pela natureza mais restritiva desse (SCHÖNING, 1988). Além disso, há indícios de que Levin atrasou a divulgação de seu trabalho buscando provar que o problema de minimização de circuitos era **NP**-completo, também sem obter sucesso (MURRAY; WILLIAMS, 2014).

Outra classe importante no estudo de complexidade é a classe **coNP**, que é o conjunto dos complementos de linguagens de **NP**. Antes de definir a classe **coNP** definimos o que é o complemento de uma linguagem: Se $L \subseteq \{0,1\}^*$, denotamos por $\overline{L}$ o complemento de L, isso é $\overline{L} = \{0,1\}^* \setminus L$. Definimos então a classe **coNP**:

Definição 2.12. (A classe **coNP**) $\mathbf{coNP} = \left\{L; \overline{L} \in \mathbf{NP}\right\}$.

É importante não confundir a classe **coNP** com o complemento da classe **NP**, até porque as duas classes possuem uma intersecção não nula, e **P** pertence a essa intersecção. Um exemplo de uma linguagem que pertence a **coNP** é a linguagem das tautologias, que são as fórmulas booleanas satisfazíveis por qualquer valoração.

Assim como a classe **NP**, é possível definir a classe **coNP** através de uma MT polinomial e de um quantificador:

Definição 2.13. (**coNP**, definição alternativa) Uma linguagem $L \subseteq \{0,1\}^*$ está em **coNP** se existe um polinômio $p : \mathbb{N} \to \mathbb{N}$ e uma MT M de tempo polinomial tal que para todo $x \in \{0,1\}^*$:

$$x \in L \leftrightarrow \forall u \in \{0,1\}^{p(|x|)}; M(x,u) = 1.$$

Também podemos definir **coNP**-completude de forma análoga à definição de **NP**-completude, uma linguagem L é **coNP**-completa se pertence a **coNP** e se toda linguagem em **coNP** é polinomialmente redutível a L. Inclusive, o problema tautologia citado anteriormente é **coNP**-completo (ARORA; BARAK, 2009).

Após definir as classes **P**, **NP** e **coNP**, é pertinente citar as relações conhecidas entre essas:

Teorema 2.3. *($\boldsymbol{P \subseteq NP \cap coNP}$)*

Em especial, se $\mathbf{P} = \mathbf{NP}$, então $\mathbf{coNP} = \mathbf{NP} = \mathbf{P}$, além disso, se $\mathbf{NP} \neq \mathbf{coNP}$, então $\mathbf{P} \neq \mathbf{NP}$. Portanto, provar resultados de complexidade em relação à classe **coNP** também pode ser um passo importante rumo à resolução da questão **P** versus **NP**.

2.5 AS CLASSES EXP E NEXP

As classes **EXP** e **NEXP** são os análogos exponenciais das classes **P** e **NP**, mais formalmente:

Definição 2.14. (A classe **EXP**) $\mathbf{EXP} = \bigcup_{c \geq 1}(\mathbf{DTIME}(2^{n^c}))$

Definição 2.15. (A classe **NEXP**) $\mathbf{NEXP} = \bigcup_{c \geq 1}(\mathbf{NTIME}(2^{n^c}))$

A classe **EXP** contém os problemas que são decidíveis em tempo exponencial, enquanto **NEXP** contém os problemas decídiveis em tempo exponencial por uma MTND, ou então, verificáveis em tempo exponencial por uma MT determinística. Devido ao fato de que essas classes contém problemas intratáveis, pode-se indagar sobre a importância de estudá-las. A questão é que as classes **EXP** e **NEXP** também capturam a essência do problema **P** versus **NP**. Se $\mathbf{EXP} \neq \mathbf{NEXP}$, então $\mathbf{P} \neq \mathbf{NP}$, e por consequência contrapositiva, se $\mathbf{P} = \mathbf{NP}$, então $\mathbf{EXP} = \mathbf{NEXP}$ (ARORA; BARAK, 2009).

Levando essa afirmação em consideração, percebemos a importância de estudar as relações entre as duas classes e outras classes de complexidade que serão definidas nesse trabalho.

2.6 COMPLEXIDADE DE ESPAÇO

Além de medir a complexidade de algoritmos através do tempo de execução desses, também é possível medir complexidade através da quantidade de espaço necessária durante a execução. Quando levamos em consideração que tempo e espaço são os principais recursos limitados que possuímos para realizar computação, percebemos a relevância de estudar as relações entre as classes definidas ao limitar cada um desses recursos. Além disso, as classes de complexidade de espaço possuem ligações importantes com a hierarquia polinomial, que será estudada na próxima seção e é central para os objetivos desse trabalho.

Antes de poder definir classes de complexidade de espaço, definimos o que é computação limitada por espaço:

Definição 2.16. (Computação limitada por espaço) Seja uma função $S : \mathbb{N} \to \mathbb{N}$ e uma linguagem $L \subseteq \{0,1\}^*$. Dizemos que $L \in \mathbf{SPACE}(S(n))$ se existe uma constante c e uma MT M que decide L tal que no máximo $c \cdot S(n)$ posições das fitas de M (com exceção da fita de entrada) são visitadas pela cabeça de M durante a computação de toda entrada de tamanho n. De maneira similar, dizemos que $L \in \mathbf{NSPACE}(S(n))$ se existe uma MTND M que decide L que nunca usa mais que $c \cdot S(n)$ posições que não contém o símbolo *branco* em entradas de tamanho n, independente de suas escolhas não determinísticas.

Uma das primeiras indagações que podem surgir diz respeito à relação da classe **DTIME**, definida anteriormente, com as classes **SPACE** e **NSPACE**. Intuitivamente podemos pensar que a restrição de tempo é mais forte que a restrição de espaço, já que espaço pode ser reutilizado enquanto tempo não. O seguinte teorema demonstra a relação conhecida entre essas classes:

Teorema 2.4. *(Relação entre classes de espaço e tempo) (ARORA; BARAK, 2009) Para toda função espaço-construtível $S : \mathbb{N} \to \mathbb{N}$:*

$$\boldsymbol{DTIME}(S(n)) \subseteq \boldsymbol{SPACE}(S(n)) \subseteq \boldsymbol{NSPACE}(S(n)) \subseteq \boldsymbol{DTIME}(2^{O(S(n))})$$

De forma análoga à complexidade de tempo, podemos definir classes de complexidade de espaço polinomial. Porém, também é possível definir classes sublineares de complexidade de espaço, pois a complexidade de espaço depende somente das fitas de trabalho e por isso a restrição de ler a entrada não limita essas classes à classes lineares ou superiores:

Definição 2.17. (Classes de complexidade de espaço)

$$\mathbf{PSPACE} = \bigcup_{c>0} \mathbf{SPACE}(n^c)$$

$$\mathbf{NPSPACE} = \bigcup_{c>0} \mathbf{NSPACE}(n^c)$$

$$\mathbf{L} = \mathbf{SPACE}(\log n)$$

$$\mathbf{NL} = \mathbf{NSPACE}(\log n)$$

A classe **PSPACE** também possui um problema completo[1], e esse problema é a

[1] A "completude" da classe PSPACE também é definida através de reduções de tempo polinomial.

linguagem das Fórmulas Booleanas Quantificadas Verdadeiras (fórmulas booleanas, não necessariamente em FNC, que possuem uma quantidade irrestrita quantificadores $\forall$ e $\exists$ alternados, verdadeiras), ou FBQV.

No caso de complexidade de espaço, a questão de espaço polinomial determinístico e não-determinístico já foi resolvida, pois a demonstração de que FQBV é **PSPACE**-completa também mostra que FBQV é **NPSPACE**-completa, dessa forma **PSPACE** = **NPSPACE** (STOCKMEYER; MEYER, 1973). Além disso, espaço não-determinístico é fechado sob complemento, diferente do que se acredita para tempo não-determinístico (através da conjectura **NP** $\neq$ **coNP**) (ARORA; BARAK, 2009).

2.7 A HIERARQUIA POLINOMIAL

A Hierarquia Polinomial, definida pela primeira vez em (MEYER; STOCKMEYER, 1972) é uma generalização das classes **NP** e **coNP**, que consiste de uma hierarquia infinita de classes de complexidade. Na área de Complexidade Computacional, há vários problemas extremamente difíceis de provar, como é o caso de mostrar que um problema é **NP**-intermediário. Por esse motivo, muitos dos resultados da área se apoiam em conjecturas, ou então afirmações que os pesquisadores acreditam ser verdadeiras. Uma dessas conjecturas é a de que a hierarquia polinomial não colapsa, que será estudada nesta seção.

Estão contidos na Hierarquia Polinomial problemas que aparentam não pertencer a **NP**, como é o caso do problema CI EXATO = $\{\langle G, k\rangle$; o maior conjunto independente de G tem tamanho $k\}$. Diferentemente do problema de decisão do Conjunto Independente em grafos, um conjunto independente não serve como certificado de que uma string pertence a CI EXATO, pois seria necessário mostrar que existe um conjunto independente de tamanho k e que todos os outros conjuntos de vértices tamanho $k+1$ não são conjuntos independentes (ARORA; BARAK, 2009). Esse problema se encaixa então na classe $\mathbf{\Sigma}_2^p$, definida a seguir:

Definição 2.18. (A Classe $\mathbf{\Sigma}_2^p$) A classe $\mathbf{\Sigma}_2^p$ é o conjunto de todas as linguagens L para as quais existe uma MT de tempo polinomial M e um polinômio q tal que:

$$x \in L \leftrightarrow \exists u \in \{0,1\}^{q(|x|)} \forall v \in \{0,1\}^{q(|x|)} M(x,u,v) = 1$$

para todo $x \in \{0,1\}^*$.

A Hierarquia Polinomial generaliza o conceito visto com a classe $\mathbf{\Sigma}_2^p$, através da

combinação de um predicado computável em tempo polinomial e de um número constante de quantificadores $\exists$ e $\forall$:

Definição 2.19. (A Hierarquia Polinomial (**PH**)) Para $i \geq 1$, uma linguagem L está em $\mathbf{\Sigma}_i^p$ se existe uma MT de tempo polinomial M e um polinômio q tal que:

$$x \in L \leftrightarrow \exists u_1 \in \{0,1\}^{q(|x|)} \forall u_2 \in \{0,1\}^{q(|x|)} \ldots Q_i u_i \in \{0,1\}^{q(|x|)} M(x, u_1, \ldots, u_i) = 1$$

para todo $x \in \{0,1\}^*$. Onde Q_i é $\forall$ se i é par e $\exists$ se i é ímpar. A Hierarquia Polinomial é o conjunto $\mathbf{PH} = \bigcup_{i>0} \mathbf{\Sigma}_i^p$.

Nota-se que segundo a definição, $\mathbf{NP} = \mathbf{\Sigma}_1^p$. Além disso, definimos $\mathbf{\Pi}_i^p = \mathbf{co\Sigma}_i^p = \left\{\overline{L}; L \in \mathbf{\Sigma}_i^p\right\}$, dessa forma $\mathbf{\Pi}_1^p = \mathbf{coNP}$. Por último, como $\mathbf{\Sigma}_i^p \subseteq \mathbf{\Pi}_{i+1}^p \subseteq \mathbf{\Sigma}_{i+2}^p$, podemos caracterizar $\mathbf{PH} = \bigcup_{i>0} \mathbf{\Pi}_i^p$ (ARORA; BARAK, 2009).

Acredita-se que $\mathbf{P} \neq \mathbf{NP}$, uma generalização dessa conjectura é a de que para todo i, $\mathbf{\Sigma}_i^p$ está estritamente contido em $\mathbf{\Sigma}_{i+1}^p$. Essa é a conjectura de que a hierarquia polinomial não colapsa (ARORA; BARAK, 2009). O teorema a seguir indica alguns pontos importantes dessa conjectura:

Teorema 2.5. *(Colapso de **PH**) (MEYER; STOCKMEYER, 1972) Para todo $i \geq 1$, se $\mathbf{\Sigma}_i^p = \mathbf{\Pi}_i^p$ então **PH** $= \mathbf{\Sigma}_i^p$, isso é, a hierarquia colapsa ao i-ésimo nível. Em especial, se **P** $=$ **NP**, então **PH** $=$ **P**, ou seja, a hierarquia polinomial colapsa a **P**.*

Na Figura 3 ilustramos a hierarquia polinomial, baseada na conjectura de que essa não colapsa e possui infinitos níveis. É importante ainda ressaltar que toda a hierarquia polinomial está contida na classe **PSPACE**.

Existem problemas completos[2] para todos os níveis da hierarquia polinomial. Em especial o caso específico do problema TBQF (com quantidade constante de quantificadores $\exists$ e $\forall$ alternados, nessa ordem), chamado de $\mathbf{\Sigma}_i$SAT é completo para o nível i da hierarquia polinomial (MEYER; STOCKMEYER, 1972). A seguir, como exemplo, a linguagem $\mathbf{\Sigma}_2$SAT:

$$\mathbf{\Sigma}_2\text{SAT} = \{\exists u_1 \forall u_2 \ \phi(u_1, u_2) = 1\}$$

onde ϕ é uma fórmula booleana não necessariamente em FNC, e u_1 e u_2 são vetores de variáveis booleanas. Nesse caso $\phi(u_1, u_2)$ indica o resultado da fórmula ϕ quando as variáveis assumem a valoração indicada nos vetores u_1 e u_2 (devidamente quantificados).

[2]Novamente, a noção de completude envolve o conceito redução polinomial e é definido de forma análoga a **NP**-completude.

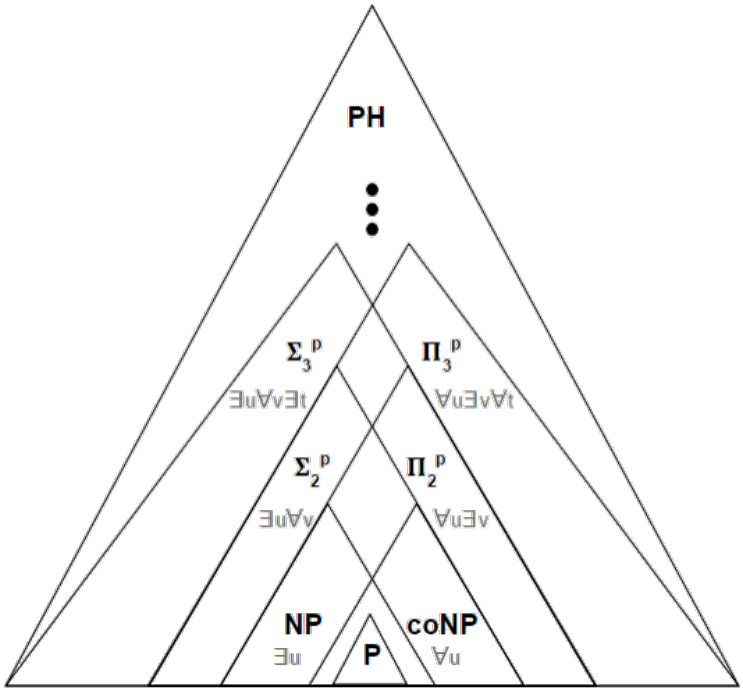

Figura 3: Hierarquia Polinomial
Fonte: Autoria própria

Também é possível definir uma linguagem $\mathbf{\Pi}_i$SAT que é completa para $\mathbf{\Pi}_i^p$ (MEYER; STOCKMEYER, 1972).

Ainda em relação ao colapso da hierarquia polinomial, pode-se afirmar que se existe problema completo para toda a hierarquia polinomial, então ela deve colapsar em algum nível. A prova é simples, se algum problema dentro de **PH** (e por consequência, dentro de um nível i de **PH**) é **PH**-completo, então qualquer problema de **PH** pode ser reduzido a este, e então todo problema de **PH** irá pertencer ao i-ésimo nível da hierarquia, colapsando-a a este.

Ainda é possível definir a hierarquia polinomial através de oráculos, pelo seguinte teorema:

Teorema 2.6. *(Hierarquia Polinomial, definição alternativa) (ARORA; BARAK, 2009) Para todo $i \geq 2$, $\mathbf{\Sigma}_i^p = \boldsymbol{NP}^{\mathbf{\Sigma}_{i-1}SAT}$, onde o lado direito da igualdade denota o conjunto de linguagens decidíveis em tempo polinomial por MTNDs com acesso ao oráculo $\mathbf{\Sigma}_{i-1}SAT$.*

Uma vez que ter acesso a um oráculo completo para determinada classe permite a uma máquina resolver qualquer problema daquela classe, alguns textos colocam como oráculo a própria classe. Nesse caso a classe $\mathbf{\Sigma}_2^p = \mathbf{NP}^{\mathrm{SAT}}$ pode ser escrita como $\mathbf{NP}^{\mathbf{NP}}$, a classe $\mathbf{\Sigma}_3^p$ como $\mathbf{NP}^{\mathbf{NP}^{\mathbf{NP}}}$ e assim por diante.

2.8 CIRCUITOS BOOLEANOS E COMPUTAÇÃO NÃO-UNIFORME

Esse capítulo estuda os circuitos Booleanos, um modelo de computação *não-uniforme*. Em contraste ao modelo padrão (ou uniforme) em que a mesma MT é utilizada para entradas de qualquer tamanho, em um modelo não-uniforme podemos utilizar algoritmos diferentes para cada tamanho de entrada. É relevante estudar esse modelo de computação pois o problema de minimização de circuitos é definido em torno de circuitos booleanos. Além disso, as classes de complexidade definidas nesta seção possuem ligações importantes com outras classes de complexidade, e inclusive com a questão **P** versus **NP**.

Um circuito Booleano é um diagrama procedural que indica como derivar uma saída de uma string binária através da aplicação de operações $\vee$ (OU lógico), $\wedge$ (E lógico) e $\neg$ (NÃO lógico). A seguir definimos formalmente o que é um circuito Booleano:

Definição 2.20. (Circuitos Booleanos) Para todo $n \in \mathbb{N}$, um circuito Booleano C de n entradas e uma saída é um grafo acíclico direcionado com n *fontes* (vértices para os quais nenhuma aresta aponta) e um *destino* (vértice do qual não parte nenhuma aresta). Todos os vértices não-destino são chamados de *gates* e possuem uma etiqueta $\vee$, $\wedge$ ou $\neg$. Os vértices com as etiquetas $\vee$ e $\wedge$ possuem *fan-in* (número de arestas que chegam no vértice) 2 e os vértices com a etiqueta $\neg$ possuem *fan-in* 1. O tamanho de C, denominado por $|C|$, é o número de vértices em C. Se C é um circuito Booleano e $x \in \{0,1\}^n$ é uma entrada, então a saída de C com x, denotada por $C(x)$, é definida através das operações dos *gates*. Mais formalmente, para cada vértice v de C associamos um valor $val(v)$, da seguinte maneira: Se v é o i-ésimo vértice fonte, então $val(v) = x_i$, e caso contrário $val(v)$ é definido recursivamente ao aplicar as operações lógicas de v nos valores dos vértices conectados a v. A saída $C(x)$ é o valor do vértice destino.

Como um circuito Booleano só aceita entradas de um determinado tamanho n, fica claro porque os circuitos Booleanos são um modelo de computação não-uniforme. Por esse motivo, não falamos em um circuito Booleano para decidir uma linguagem $L \subseteq \{0,1\}^*$, e sim em *famílias de circuitos*:

Definição 2.21. (Famílias de circuitos e reconhecimento de linguagens) Seja $T : \mathbb{N} \to \mathbb{N}$ uma função. Uma família de circuitos de tamanho $T(n)$ é uma sequência $\{C_n\}_{n\in\mathbb{N}}$ de circuitos Booleanos, onde C_n possui n entradas e uma saída, e $|C_n| \leq T(n)$ para todo n. Dizemos que uma linguagem $L \subseteq \{0,1\}^*$ pertence a $\mathbf{SIZE}(T(n))$ se existe uma família de circuitos $\{C_n\}_{n\in\mathbb{N}}$ de tamanho $T(n)$ tal que para todo $x \in \{0,1\}^n$, $x \in L \leftrightarrow C_n(x) = 1$.

Como existem circuitos Booleanos de tamanho $O(2^n/n)$ para todas as funções booleanas de $\{0,1\}^n$ para $\{0,1\}$ (SHANNON, 1949b), é interessante focar a discussão em circuitos polinomiais:

Definição 2.22. (A classe $\mathbf{P_{/poly}}$) (KARP; LIPTON, 1982) $\mathbf{P_{/poly}}$ é a classe de linguagens decidíveis por famílias de circuitos de tamanho polinomial. Isso é, $\mathbf{P_{/poly}} = \bigcup_{c>0} \mathbf{SIZE}(n^c)$.

É possível realizar a mesma objeção em relação à praticidade da classe $\mathbf{P_{/poly}}$ que se faz em relação à classe $\mathbf{P}$, no sentido de que circuitos de tamanho n^{100} não são necessariamente pequenos. O mesmo argumento utilizado para a classe $\mathbf{P}$ pode ser usado nesse caso, pois acredita-se que $\mathbf{NP} \not\subseteq \mathbf{P_{/poly}}$ (ARORA; BARAK, 2009).

O teorema a seguir esclarece a relação entre as classes $\mathbf{P}$ e $\mathbf{P_{/poly}}$:

Teorema 2.7. *(Relação entre $\boldsymbol{P}$ e $\boldsymbol{P_{/poly}}$) (ARORA; BARAK, 2009)* $\boldsymbol{P \subseteq P_{/poly}}$

É interessante notar que a inclusão do teorema anterior é estrita, pois qualquer linguagem unária pertence à classe $\mathbf{P_{/poly}}$, e existem linguagens unárias que são indecidíveis:

Afirmação 2.3. *(Linguagens unárias e a classe $\boldsymbol{P_{/poly}}$) Se L é uma linguagem unária, então $L \in \boldsymbol{P_{/poly}}$.*

A demonstração da afirmação é bem simples e envolve a seguinte ideia: se uma string $x \in \{0,1\}^n$ pertence a L, então descrevemos o circuito C_n que aceita a string como sendo o E lógico de todas as entradas (negadas se a string é unária com caracteres 0), e se a string não pertence a L, então o circuito é o circuito canônico que responde 0 para qualquer string.

Um exemplo, então, de linguagem unária indecidível é a linguagem UHALT = $\{1^n;$ a expansão binária de n representa o par $\langle M, x\rangle$ tal que M para com $x\}$. A ideia é que só precisa *existir* uma família de circuitos, mesmo que não seja possível *construí-la* (como é o caso da linguagem UHALT).

Para evitar esse problema, poderíamos restringir a classe $\mathbf{P_{/poly}}$ aos circuitos que podem ser construídos em tempo polinomial por uma MT (circuitos $\mathbf{P}$-uniformes), porém realizar essa restrição colapsa a classe $\mathbf{P_{/poly}}$ à classe $\mathbf{P}$ (ARORA; BARAK, 2009).

Da mesma forma que outras classes de complexidade, podemos definir de maneira alternativa a classe $\mathbf{P_{/poly}}$, através de máquinas de Turing que recebem *conselhos*:

Definição 2.23. (MT que recebe conselhos) Sejam $T, a : \mathbb{N} \to \mathbb{N}$ funções. A classe de linguagens decidíveis por MTs de tempo $T(n)$ que recebem $a(n)$ bits de conselho, denotada por $\mathbf{DTIME}(T(n))/a(n)$, contém toda linguagem $L \subseteq \{0,1\}^*$ tal que existem uma sequência $\{\alpha_n\}_{n\in\mathbb{N}}$ de strings com $\alpha_n \in \{0,1\}^{a(n)}$ e uma MT M que satisfazem:

$$M(x, \alpha_n) = 1 \leftrightarrow x \in L$$

para todo $x \in \{0,1\}^n$, onde na entrada (x, α_n) a máquina M roda em tempo $T(n)$.

Teorema 2.8. *(MTs de tempo polinomial que recebem conselhos decidem $\boldsymbol{P_{/poly}}$) (KARP; LIPTON, 1982)* $\boldsymbol{P_{/poly}} = \bigcup_{c,d} \boldsymbol{DTIME}(n^c)/n^d$.

A questão que surge então é: será que a classe **NP** pode estar contida em $\mathbf{P_{/poly}}$? Certamente uma resposta positiva seria interessante, pois tornaria os problemas **NP**-completos mais tratáveis (pelo menos até determinado tamanho), e uma resposta negativa também seria muito importante, pois implicaria que $\mathbf{P} \neq \mathbf{NP}$. A resposta encontrada até o momento para essa pergunta é não, porém assumindo que a hierarquia polinomial não colapsa:

Teorema 2.9. *(Teorema Karp-Lipton) (KARP; LIPTON, 1982) Se $\boldsymbol{NP} \subseteq \boldsymbol{P_{/poly}}$, então $\boldsymbol{PH} = \boldsymbol{\Sigma}_2^p$.*

E o seguinte teorema também indica que $\mathbf{P_{/poly}}$ provavelmente não contém a classe **EXP**:

Teorema 2.10. *(Teorema de Meyer) (KARP; LIPTON, 1982) Se $\boldsymbol{EXP} \subseteq \boldsymbol{P_{/poly}}$, então $\boldsymbol{EXP} = \boldsymbol{\Sigma}_2^p$.*

Os teoremas anteriores indicam que provavelmente a classe **NP** não está contida em $\mathbf{P_{/poly}}$, porém também é importante observar que para qualquer n, existem funções $f : \{0,1\}^n \to \{0,1\}$ que não são computáveis por circuitos polinomiais (SHANNON, 1949b). Inclusive, uma prova alternativa para esse fato mostra que a *maioria* dessas funções não possuem circuitos polinomiais (ARORA; BARAK, 2009). Encontrar apenas uma dessas funções que não pertença a **NP** seria suficiente para provar $\mathbf{NP} \not\subseteq \mathbf{P_{/poly}}$, e como consequência, que $\mathbf{P} \neq \mathbf{NP}$.

3 DIAGONALIZAÇÃO E O TEOREMA DE LADNER

A técnica de diagonalização foi criada por Georg Cantor no século 19, sendo utilizada posteriormente em muitos resultados matemáticos e computacionais, e inclusive por Alan Turing para provar que o problema da parada[1] é indecidível (ou seja, que não existe algoritmo que resolve esse problema) (TURING, 1936). A técnica de diagonalização é utilizada para mostrar que, se $\mathbf{P} \neq \mathbf{NP}$, então existem problemas **NP**-intermediários (LADNER, 1975), e por esse motivo é central para os objetivos desse trabalho.

A técnica é muito utilizada na área de Complexidade Computacional para mostrar que duas classes de complexidade são diferentes (o que é feito mostrando uma máquina de uma das classes que difere de todas as outras de outra classe, no sentido de que suas respostas são diferentes em pelo menos uma entrada).

Um teorema muito importante para a área de Complexidade Computacional (cuja prova envolve a técnica de diagonalização) é o Teorema de hierarquia de tempo determinístico, que indica que permitir à máquinas de Turing determinísticas mais tempo de execução aumenta o conjunto de linguagens que essas podem decidir:

Teorema 3.1. *(A Hierarquia de Tempo Determinística) (HARTMANIS; STEARNS, 1966) Sejam f e g funções tempo-construtíveis que satisfaçam $f(n)\log f(n) = o(g(n))$, então:*

$$\boldsymbol{DTIME}(f(n)) \subsetneq \boldsymbol{DTIME}(g(n))$$

Demonstração. Para mostrar a ideia principal da prova, provamos o resultado mais simples $\mathbf{DTIME}(n) \subsetneq \mathbf{DTIME}(n^{1,5})$:

Considere a seguinte MT D: "Na entrada x, execute por $|x|^{1,4}$ passos a MT Universal U do Teorema 2.1 com a entrada $\langle x, x\rangle$, de forma a simular a execução de M_x com a entrada x. Se U tem como saída um bit $b \in \{0,1\}$ nesse tempo, responda $1-b$,

[1] O problema da parada é o seguinte: Dados a descrição de uma máquina M_α através da string α e uma entrada $x \in \{0,1\}^*$, o problema da parada envolve decidir se a máquina M_α rodando com a string x para.

caso contrário responda 0".

Por definição D para em até $n^{1,4}$ passos e a linguagem L decidida por D está em $\mathbf{DTIME}(n^{1,5})$. Basta então provar que $L \notin \mathbf{DTIME}(n)$. Assumimos com o intuito de derivar uma contradição que existe uma MT M e uma constante c tal que M, dada uma entrada $x \in \{0,1\}^*$, para em até $c|x|$ passos e tem como saída $D(x)$.

O número de passos necessários para simular M através da MT Universal U em qualquer entrada x é de no máximo $c'c|x|\log|x|$ para uma constante c' que depende apenas de M. Existe uma constante n_0 tal que $n^{1,4} > c'cn\log n$ para todo $n \geq n_0$. Seja x a string de tamanho maior ou igual a n_0 que representa M. Então, $D(x)$ irá obter a saída $b = M(x)$ em $|x|^{1,4}$ passos, porém pela definição de D, temos que $D(x) = 1 - b \neq M(x)$. Derivamos uma contradição.

□

Em especial, devido à esse teorema sabe-se que $\mathbf{P} \subsetneq \mathbf{EXP}$, pois qualquer função do tipo n^c é $o(2^{n^c})$, o que nos leva a acreditar que pelo menos uma das inclusões $\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{EXP}$ seja estrita. De fato, muitos pesquisadores acreditam que ambas as inclusões são estritas (ARORA; BARAK, 2009).

Também existe um Teorema de Hierarquia de Tempo Não-determinístico, que indica que máquinas de Turing não-determinísticas também podem reconhecer um conjunto maior de linguagens quando podem executar por mais tempo:

Teorema 3.2. *(A Hierarquia de Tempo Não-determinístico) (COOK, 1973) Sejam f e g funções tempo-construtíveis que satisfaçam $f(n+1) = o(g(n))$, então:*

$$\boldsymbol{NTIME}(f(n)) \subsetneq \boldsymbol{NTIME}(g(n))$$

Da mesma maneira que o teorema para máquinas de Turing determinísticas, o teorema anterior implica que $\mathbf{NP} \subsetneq \mathbf{NEXP}$.

Interessantemente, as classes de complexidade de espaço também possuem um teorema de hierarquia, ou seja, máquinas de Turing com acesso a mais posições em suas fitas de trabalho podem decidir mais linguagens, como mostra o teorema a seguir:

Teorema 3.3. *(Hierarquia de espaço) (STEARNS et al., 1965) Sejam f e g funções espaço-construtíveis que satisfazem $f(n) = o(g(n))$, então:*

$$\boldsymbol{SPACE}(f(n)) \subsetneq \boldsymbol{SPACE}(g(n))$$

Percebe-se que nesse teorema não há a presença do fator logarítmico do Teorema 3.1. Isso se deve ao fato de que a prova dos dois é muito parecida e ambas se utilizam de máquinas de Turing Universais, a diferença é que uma MT Universal possui *overhead* logarítmico de tempo, porém apenas constante de espaço (ARORA; BARAK, 2009).

A técnica de Diagonalização nos permite provar diversas separações de classes de complexidade. A Figura 4 ilustra as relações de pertinência entre várias das classes definidas até o momento (ARORA; BARAK, 2009).

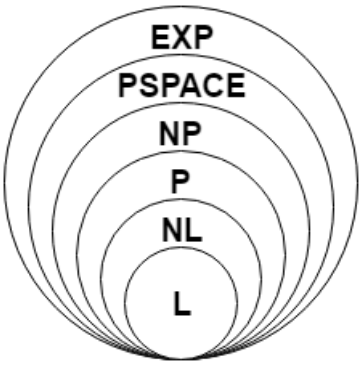

Figura 4: Relação entre classes de complexidade definidas.
Fonte: Autoria própria.

Em relação à Figura 4, podem ser observadas as seguintes relações:

$$\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXP}.$$

Pelo teorema de hierarquia de tempo $\mathbf{P} \subsetneq \mathbf{EXP}$, e além disso, pelo teorema de hierarquia de espaço $\mathbf{L} \subsetneq \mathbf{PSPACE}$, o que indica que pelo menos algumas das inclusões acima são estritas. A maioria dos pesquisadores acredita que todas as inclusões são estritas (ARORA; BARAK, 2009).

Outro resultado muito importante para o estudo de Complexidade Computacional em geral, e em especial para os objetivos deste trabalho, é o Teorema de Ladner. A prova mostrada abaixo é baseada em (ARORA; BARAK, 2009):

Teorema 3.4. *(Teorema de Ladner, ou a existência de linguagens **NP**-intermediárias) (LADNER, 1975) Suponha que $\boldsymbol{P \neq NP}$. Então existe uma linguagem $L \in \boldsymbol{NP} \setminus \boldsymbol{P}$ que não é **NP**-completa.*

Demonstração. Para qualquer função $H : \mathbb{N} \to \mathbb{N}$, definimos a linguagem SAT_H que contém todas as fórmulas booleanas de tamanho n com um *enchimento* de $n^{H(n)}$ caracteres

1. Ou seja, $\text{SAT}_H = \{\psi 01^{n^{H(n)}}; \psi \in \text{SAT} \text{ e } |\psi| = n\}$.

Definimos a função $H : \mathbb{N} \to \mathbb{N}$ da seguinte maneira: $H(n)$ é o menor número $i < \log\log n$ tal que para todo $x \in \{0,1\}^*$ com $|x| \leq \log n$, M_i tem como saída $\text{SAT}_H(x)$ em até $i|x|^i$ passos. Se não existe esse número i, então $H(n) = \log\log n$. Para computar $H(n)$ precisamos (1) computar $H(i)$ para todo $i \leq \log n$, (2) simular no máximo $\log\log n$ máquinas em entradas de tamanho $\log n$ por menos de $\log\log n(\log n)^{\log\log n} = o(n)$ passos e (3) computar SAT para entradas de tamanho no máximo $\log n$, então se $T(n)$ denota o tempo para computar $H(n)$, $T(n) \leq \log T(\log n) + O(n^2)$, portanto $H(n)$ é computável em tempo $O(n^2)$. A função H foi definida dessa maneira para garantir a seguinte afirmação:

Afirmação 3.1. *$SAT_H \in \boldsymbol{P} \leftrightarrow H(n) = O(1)$. Além disso, se $SAT_H \notin \boldsymbol{P}$ então $H(n)$ tende ao infinito junto a n.*

Demonstração. ($\text{SAT}_H \in \mathbf{P} \to H(n) = O(1)$) : Suponha que há uma MT que decide SAT_H em cn^c passos. Como M é representada por infinitas strings distintas, existe um número $i > c$ tal que $M = M_i$. A própria definição de $H(n)$ então implica que para $n > 2^{2^i}$, $H(n) \leq i$, portanto $H(n) = O(1)$.

($H(n) = O(1) \to \text{SAT}_H \in \mathbf{P}$) : Se $H(n) = O(1)$, $H(n)$ pode ser um número finito de valores, e em especial, existe um i tal que $H(n) = i$ para uma quantidade infinita de valores n. Porém isso implica que a máquina M_i decide SAT_H em tempo in^i, pois caso exista uma entrada x na qual M_i falha em responder corretamente dentro desse limite de tempo, então para todo $n > 2^{|x|}$ teríamos $H(n) \neq i$. Note que esse argumento se mantém se apenas assumirmos que para uma constante C, $H(n) \leq C$ para uma quantidade infinita de n's, o que prova que se $\text{SAT}_H \notin \mathbf{P}$ então $H(n)$ tende ao infinito junto a n. □ (afirmação)

Usando essa afirmação podemos então provar que se $\mathbf{P} \neq \mathbf{NP}$ então SAT_H não esta nem em $\mathbf{P}$ e nem é **NP**-completo:

Suponha que $\text{SAT}_H \in \mathbf{P}$, então pela afirmação acima $H(n) \leq C$ para alguma constante C, o que indica que SAT_H nada mais é que a linguagem SAT *enchida* com no máximo uma quantidade polinomial de 1's (no caso, n^C). Porém nesse caso um algoritmo polinomial para SAT_H poderia ser usado para resolver SAT em tempo polinomial, implicando que $\mathbf{P} = \mathbf{NP}$ e violando a hipótese inicial. Suponha então que SAT_H é **NP**-completo. Isso significa que há uma redução f de SAT para SAT_H que roda em tempo $O(n^i)$ para alguma constante i. Já que concluímos que SAT_H não pertence a $\mathbf{P}$, a afirmação acima implica que $H(n)$ tende ao infinito. Já que a redução funciona em tempo $O(n^i)$ apenas, para n suficientemente grande essa deve mapear instâncias SAT de

tamanho n para instâncias SAT_H de tamanho menor que $n^{H(n)}$. Portanto para fórmulas suficientemente grandes ϕ, a redução f deve mapeá-las a strings do tipo $\psi 01^{n^{H(|\psi|)}}$, onde ψ é menor que ϕ por um fator polinomial fixo. Porém a existência dessa redução nos dá um algoritmo[2] polinomial para SAT, que contradiz a hipótese $\mathbf{P} \neq \mathbf{NP}$. □

A Figura 5 ilustra a relação entre as classes de complexidade **P** e **NP** caso sejam diferentes, de acordo com o Teorema de Ladner (os problemas **NP**-intermediários estariam contidos na região de **NP** que não está em **P** nem em **NP**-completo). Apesar do teorema de Ladner mostrar que existe uma linguagem (SAT_H) que nem pertence a classe **P**, nem é **NP**-completa (assumindo que $\mathbf{P} \neq \mathbf{NP}$), essa linguagem aparenta ser muito artificial, e até hoje não foram encontradas linguagens naturais que se encaixem na denominação **NP**-intermediária (ARORA; BARAK, 2009). Mesmo assim vários problemas, como isomorfismo de grafos, resíduo quadrático e logaritmo discreto, são candidatos a pertencerem à essa classe.

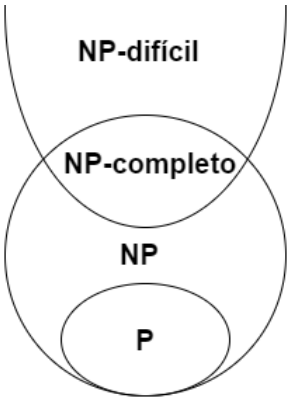

Figura 5: Relação entre P e NP caso as classes sejam diferentes.
Fonte: Autoria própria.

Apesar dos vários teoremas apresentados, a técnica de diagonalização apresenta alguns limites muito importantes. Para poder estudar esses limites, antes é necessário entender explicitamente o que classifica uma prova por diagonalização. Arora e Barak (2009) indicam que diagonalização é qualquer técnica que depende de apenas duas propriedades de MTs:

1. A existência de uma representação eficiente de MTs por strings.

[2] O algoritmo recursivo, denominado A, é o seguinte: Assuma o número N como o valor tal que $H(n) > i$ para $n > N$ (lembrar que a redução f roda em tempo $O(n^i)$). Ao receber como entrada uma fórmula ϕ, se $|\phi| \leq N$, então compute o resultado por força bruta, caso contrário compute $x = f(\phi)$, se x não é da forma $\psi 01^{n^{H(|\psi|)}}$, então retorne FALSO, caso contrário, retorne $A(\psi)$.

2. A habilidade de uma MT poder simular qualquer outra sem que haja muito *overhead* de espaço ou tempo.

A questão é que qualquer técnica que utilize somente essas duas propriedades considera as máquinas de Turing como *caixas pretas*, e como máquinas de Turing com oráculo também obedecem à ambas as propriedades (ARORA; BARAK, 2009), qualquer resultado de MTs ou complexidade que seja provado através de diagonalização também é válido para máquinas de turing com acesso ao oráculo O, independente de qual é o oráculo O. Esses resultados são chamados de resultados *relativizantes*, e em especial, seja $\mathbf{P} = \mathbf{NP}$ ou $\mathbf{P} \neq \mathbf{NP}$ verdadeiro, esse resultado não é relativizante, devido ao seguinte teorema (que curiosamente é provado através de diagonalização):

Teorema 3.5. *(Existem oráculos A e B tal que $\boldsymbol{P}^A = \boldsymbol{NP}^A$ e $\boldsymbol{P}^B \neq \boldsymbol{NP}^B$) (BAKER et al., 1975)*

Demonstração. O oráculo A é o oráculo para a seguinte linguagem EXPCOM:

$$\text{EXPCOM} = \{\langle M, x, 1^n\rangle ; M \text{ responde 1 em até } 2^n \text{ passos}\}$$

É fácil ver que um oráculo para EXPCOM permite realizar uma computação exponencial em apenas um passo computacional, portanto $\mathbf{EXP} \subseteq \mathbf{P}^{\text{EXPCOM}}$. Por outro lado, podemos simular a execução de uma MTND M com oráculo para EXPCOM em tempo exponencial, pois podemos enumerar todas as escolhas não determinísticas de M e responder à todas as consultas ao oráculo EXPCOM dentro desse limite de tempo. Como consequência $\mathbf{P}^{\text{EXPCOM}} = \mathbf{NP}^{\text{EXPCOM}} = \mathbf{EXP}$.

O oráculo B requer uma construção mais complexa. Para qualquer linguagem B, definimos a linguagem U_B:

$$U_B = \{1^n; \text{alguma string de tamanho } n \text{ pertence a } B\}$$

Para qualquer oráculo B, a linguagem U_B certamente está em $\mathbf{NP}^{\text{B}}$, pois uma MTND pode adivinhar não-deterministicamente uma string $x \in \{0,1\}^n$ tal que $x \in B$ e então consultar o oráculo B em relação à pertinência dessa string. Agora construímos um oráculo (ou linguagem) B tal que $\mathbf{P}^B \neq \mathbf{NP}^B$.

Construção de B: Para todo i, temos que M_i é a MT com oráculo representada pela expansão binária de i. Construímos a linguagem B em estágios, e no estágio i garantimos que M_i^B não decide a linguagem U_B em tempo $2^n/10$. Inicialmente a linguagem

B é vazia, e gradualmente adicionaremos strings a B. Cada estágio determina o estado (isso é, se estarão ou não em B) de um número finito de strings.

Estágio i: Até o momento, declaramos se um número finito de strings está ou não em B. Escolha n grande o suficiente para que exceda o tamanho de todas as strings que pertencem a B, e rode M_i na entrada 1^n por $2^n/10$ passos. Sempre que M_i consultar o oráculo em relação à strings que já definimos se estão ou não em B, respondemos corretamente. Porém se M_B consultar o oráculo em relação à uma string cujo estado não foi definido, declaramos então que a string não pertence a B. É importante notar que até o momento não declaramos que B possui uma string de tamanho n. Agora é o momento de garantir que se M_i para em até $2^n/10$ passos então sua resposta (qualquer que seja) para a string 1^n é incorreta. Um ponto importante é que decidimos no máximo o estado de $2^n/10$ strings em $\{0,1\}^n$, e todas foram declaradas como não pertencentes a B. Se M_i aceita a string 1^n, declaramos que todas as strings de tamanho n não pertencem a B, garantindo que $1^n \notin U_B$. Pelo outro lado, se M_i rejeita a string 1^n, escolhemos qualquer outra string x de tamanho n que M_i não tenha consultado (essa string existe pois M_i consultou no máximo $2^n/10$ strings de tamanho n), e declaramos que x pertence a B. Garantindo que $1^n \in U_B$. Em qualquer caso, a máquina M_i erra sua resposta.

Devido aos fatos de que todo polinômio $p(n)$ é menor que $2^n/10$ para n suficientemente grande e de que toda MT M é representada por uma quantidade infinita de strings, nossa construção garante que nenhuma MT M de tempo polinomial decide U_B. Dessa forma, mostramos que $U_B \notin \mathbf{P}^B$ e como consequência $\mathbf{P}^B \neq \mathbf{NP}^B$. □

Esse teorema nos mostra que não é possível resolver a questão **P** versus **NP** através apenas de técnicas de diagonalização[3] (ou técnicas *relativizantes*). É nesse momento que se mostra a importância do fato de que a computação é *local* (isso é, uma MT analisa e modifica apenas uma parcela constante de suas fitas a cada passo computacional), pois esse resultado não é um resultado *relativizante*, e provavelmente será muito importante para resolver a questão **P** versus **NP** (ARORA; BARAK, 2009).

[3]Pois se fosse possível, o resultado encontrado deveria se manter com qualquer oráculo O, o que vimos que não acontece.

4 COMPLEXIDADE COMPUTACIONAL E ALEATORIEDADE

Neste capítulo são explorados modelos de computação que fazem uso de escolhas aleatórias sob o ponto de vista da Complexidade Computacional. Os conceitos e classes definidos Neste capítulo são diretamente usados nos algoritmos apresentados para problemas candidatos a **NP**-intermediários. Novamente, a menos que indicado o contrário, as definições e notações apresentadas são as mesmas de (ARORA; BARAK, 2009).

Antes, porém, é feita uma pequena revisão de alguns conceitos de Probabilidade Discreta que serão importantes nas definições e teoremas desse capítulo.

4.1 PROBABILIDADE DISCRETA

Um *espaço de probabilidade finito* é um conjunto $\Omega = \{\omega_1, \omega_2, \ldots, \omega_N\}$ junto de um conjunto de números $p_1, p_2, \ldots\ p_N \in [0, 1]$ tal que $\sum_{i=1}^{N} p_i = 1$. Um elemento aleatório é selecionado desse espaço ao escolher ω_i com probabilidade p_i. Se x é escolhido do espaço de probabilidade Ω, então denotamos isso por $x \in_R \Omega$. Se nenhuma distribuição é definida, então é utilizada a distribuição uniforme entre os elementos de Ω ($p_i = 1/N$ para todo i).

Um *evento* sobre o espaço Ω é um subconjunto $A \subseteq \Omega$ e a *probabilidade* de A ocorrer, denominada $\Pr[A]$ é igual a $\sum_{i;\omega_i \in A} p_i$. A seguinte desigualdade em relação à probabilidade da união de eventos é conhecida como *Union Bound*: para todo conjunto de eventos $A_1, A_2, \ldots, A_n$:

$$\Pr[\cup_{i=1}^{n} A_i] \leq \sum_{i=1}^{n} \Pr[A_i]$$

Uma *variável aleatória* é um mapeamento de um espaço de probabilidade para o conjunto $\mathbb{R}$. Por exemplo, se Ω é o espaço de resultados possíveis do lançamento de n moedas, então poderíamos definir uma variável aleatória X como a quantidade de moedas que caiu com a face "cara" para cima. Caso uma variável aleatória possa assumir somente os valores 0 e 1 dizemos que ela é uma variável aleatória indicadora. A *esperança* de uma variável aleatória X, denotada por $\mathrm{E}[X]$, é a média ponderada dos valores de X. Isso é,

$\mathrm{E}[X] = \sum_{i=1}^{N} p_i X(\omega_i)$. A seguinte afirmação segue da definição de $\mathrm{E}[X]$:

Afirmação 4.1. *(Linearidade da Esperança) Para X, Y variáveis aleatórias sobre o espaço Ω, denotamos por $X + Y$ a variável aleatória que mapeia ω para $X(\omega) + Y(\omega)$. Então:*

$$E[X + Y] = E[X] + E[Y]$$

Ainda, para um número real α e uma variável aleatória X, definimos αX como a variável aleatória que mapeia $\alpha \cdot X(\omega)$. Note que $\mathrm{E}[\alpha X] = \alpha \mathrm{E}[X]$.

Também em relação à esperança de uma variável aleatória X temos o seguinte lema, conhecido como *desigualdade de Markov*, que será útil em algumas demonstrações:

Lema 4.1. *(Desigualdade de Markov) Qualquer variável aleatória não negativa X satisfaz:*

$$Pr(X \geq k \cdot E[X]) \leq \frac{1}{k}$$

Outro lema importante em relação à concentração da soma de variáveis aleatórias indicadoras independentes, que encontra várias aplicações na área de complexidade computacional, é o *limitante de Chernoff*:

Lema 4.2. *Sejam $X_1, X_2, \ldots, X_n$ variáveis aleatórias indicadoras (que só assumem valores em $\{0, 1\}$) mutuamente independentes , e seja $\mu = \sum_{i=1}^{n} E[X_i]$, então para todo $\delta > 0$:*

$$\Pr\left[\sum_{i=1}^{n} X_i \geq (1+\delta)\mu\right] \leq \left[\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right]^{\mu}$$

$$\Pr\left[\sum_{i=1}^{n} X_i \leq (1-\delta)\mu\right] \leq \left[\frac{e^{-\delta}}{(1-\delta)^{(1-\delta)}}\right]^{\mu}$$

Combinando as duas desigualdades do limitante de Chernoff, obtemos a seguinte fórmula:

$$\Pr\left[\left|\sum_{i=1}^{n} X_i - \mu\right| \geq c\mu\right] \leq 2 \cdot e^{-min\{c^2/4, c/2\}\mu}$$

Por último, ainda definimos a distância estatística entre duas variáveis aleatórias, um conceito que será importante em algumas seções desse capítulo:

Definição 4.1. (Distância Estatística) Seja Ω um espaço de probabilidade finito, definimos a distância estatística entre duas variáveis aleatórias X e Y que mapeiam elementos de Ω para elementos em algum conjunto Θ, como:

$$\Delta(X, Y) = \max_{S \subseteq \Theta} |\Pr[X \in S] - \Pr[Y \in S]|$$

Caso $\Delta(X, Y) = 0$, dizemos que ambas são igualmente distribuídas sobre Θ.

4.2 COMPUTAÇÃO ALEATORIZADA

Até o momento o modelo de computação analisado, a máquina de Turing, não faz uso de um aspecto da nossa realidade: a habilidade de fazer *escolhas aleatórias*. Ainda há muita discussão em relação à existência de aleatoriedade verdadeira no mundo, porém quando jogamos uma moeda pra cima o resultado aparenta ser suficientemente aleatório para qualquer efeito prático. Nesse contexto, faz sentido pensar em máquinas de Turing que possuem a habilidade de jogar moedas e tomar decisões baseadas no resultado, essas são as Máquinas de Turing Probabilísticas:

Definição 4.2. (Máquina de Turing Probabilística) Uma Máquina de Turing Probabilística (MTP) é uma MT com duas funções de transição δ_0 e δ_1. Para executar uma MTP M com uma entrada x, escolhemos em cada passo computacional, com probabilidade $1/2$ aplicar a função de transição δ_0, e com probabilidade $1/2$ aplicar a função de transição δ_1. Cada escolha é feita independentemente das escolhas anteriores. A máquina só responde 1 (aceita) ou 0 (rejeita). Denotamos por $M(x)$ a variável aleatória que corresponde ao valor que M responde no final desse processo. Para uma função $T : \mathbb{N} \to \mathbb{N}$, dizemos que M roda em tempo $T(n)$ se, para qualquer entrada $x \in \{0,1\}^*$, M rodando com x para em até $T(|x|)$ passos, independentemente de suas escolhas aleatórias.

É interessante notar que uma MTP se assemelha a uma MTND, no sentido de que ambas possuem duas funções de transição, porém MTPs foram pensadas com o intuito de modelar computação realística. A principal diferença entre os dois modelos diz respeito à interpretação do grafo de computação possíveis, enquanto uma MTND aceita uma string se nesse grafo *existe* um caminho que aceita, em uma MTP estamos preocupados com a *fração* dos caminhos que aceitam, que é outra maneira de definir a probabilidade de aceitação $\Pr[M(x) = 1]$.

Definimos então a classe **BPP**, que visa capturar computação probabilística eficiente. Abaixo, para uma linguagem $L \subseteq \{0,1\}^*$ e string $x \in \{0,1\}^*$, definimos $L(x) = 1$ se $x \in L$ e $L(x) = 0$ caso contrário:

Definição 4.3. (As classes **BPTIME** e **BPP**) (GILL, 1974) Para $T : \mathbb{N} \to \mathbb{N}$ e $L \subseteq \{0,1\}^*$, dizemos que uma MTP M decide L em tempo $T(n)$ se para todo $x \in \{0,1\}^*$, M ro-

dando com x para em até $T(|x|)$ passos independentemente das escolhas aleatórias que realiza e $\Pr[M(x) = L(x)] \geq 2/3$. Definimos então $\mathbf{BPTIME}(T(n))$ como a classe das linguagens decidíveis por MTPs em tempo $O(T(n))$ e definimos $\mathbf{BPP} = \bigcup_{c>0} \mathbf{BPTIME}(n^c)$.

Pode-se pensar que a probabilidade $2/3$ de acerto necessária para que uma máquina pertença a **BPP** é muito arbitrária, e de certa forma é, mas pelo motivo de que podemos trocar essa probabilidade por qualquer outra acima de $1/2$ sem mudar a classe **BPP**, e até mesmo por $1/2 + n^{-c}$ para uma constante $c > 0$. Isso se deve ao fato de que podemos transformar uma MTP de tempo polinomial M com probabilidade de sucesso maior ou igual a $1/2 + n^{-c}$ em uma MTP de tempo polinomial M' com probabilidade de sucesso exponencialmente próxima de 1, de acordo com o seguinte teorema:

Teorema 4.1. *(Redução de erro para **BPP**) (ARORA; BARAK, 2009) Seja $L \subseteq \{0,1\}^*$ uma linguagem e suponha que existe uma MTP de tempo polinomial M tal que para todo $x \in \{0,1\}^*$, $Pr[M(x) = L(x)] \geq 1/2 + |x|^{-c}$. Então para toda constante $d > 0$ existe uma MTP M' de tempo polinomial tal que para todo $x \in \{0,1\}^*$, $Pr[M'(x) = L(x)] \geq 1 - 2^{-|x|^d}$.*

Demonstração. O comportamente de M' é o seguinte: para toda entrada $x \in \{0,1\}^*$, execute $M(x)$ $k = 8|x|^{2c+d}$ vezes obtendo k saídas $y_1, \ldots, y_k \in \{0,1\}$. A resposta de M' é a resposta que mais aparece dentre 0 e 1.

Para analisar essa MTP, definimos a variável aleatória X_i que é igual a 1 se $y_i = L(x)$ e igual a 0 caso contrário. Note que $X_1, \ldots, X_k$ são variáveis aleatórias indicadoras independentes com $E[X_i] = \Pr[X_i = 1] \geq p$ para $p = 1/2 + |x|^{-c}$. Pelo limitante de Chernoff e para δ suficientemente pequeno, temos o seguinte:

$$\Pr\left[|\sum_{i=1}^{k} X_i - pk| \geq \delta pk\right] \leq e^{-(\delta/4)pk}$$

Nesse caso $p = 1/2 + |x|^{-c}$ e fazendo $\delta = |x|^{-c}/2$ garante que se $\sum_{i=1}^{k} X_i \geq pk - \delta pk$ então a máquina M' responde corretamente. Portanto a probabilidade de que M' erre sua resposta possui o seguinte limite superior:

$$e^{-(1/4|x|^{2c})(1/2)8|x|^{2c+d}} \leq 2^{-|x|^d}$$

□

Sem que hajam alterações na classe **BPP**, ainda podemos utilizar moedas "injustas" (que possuem probabilidade de cara diferente de $1/2$) ou permitindo à maquina

rodar em tempo *esperado* polinomial (ARORA; BARAK, 2009).

Também é importante ressaltar que apesar de uma MTP ter probabilidade de errar a resposta em determinados momentos, essa probabilidade depende apenas das escolhas aleatórias dessa, e não da string em particular. Dessa forma a máquina possui a mesma probabilidade de errar para qualquer string x.

Assim como outras classes de complexidade, a classe **BPP** possui uma definição alternativa, que é útil em algumas demonstrações de teoremas. Nesse caso, a definição alternativa envolve uma MT normal que recebe como entrada adicional toda a sequência de "lançamentos de moeda" (na forma de uma string binária) aleatórios que são necessários para a execução de uma MTP:

Definição 4.4. (**BPP**, definição alternativa) (GILL, 1974) Uma linuagem $L \subseteq \{0,1\}^*$ está em **BPP** se existe uma MT polinomial M e um polinômio $p : \mathbb{N} \to \mathbb{N}$ tal que para todo $x \in \{0,1\}^*$, $Pr_{r \in_R \{0,1\}^{p(|x|)}}[M(x,r) = L(x)] \geq 2/3$.

Pela definição de **BPP** podemos deduzir que $\mathbf{P} \subseteq \mathbf{BPP}$, pois uma MT é um caso especial de uma MTP em que as duas funções de transição δ_0 e δ_1 são iguais. Além disso, pela definição alternativa, percebemos que $\mathbf{BPP} \subseteq \mathbf{EXP}$ pois em tempo $O(2^{poly(n)})$ podemos enumerar todas as escolhas aleatórias possíveis de uma MTP.

Um outro problema central de Complexidade Computacional é se $\mathbf{BPP} = \mathbf{P}$, e interessantemente a maioria dos pesquisadores acredita que sim, ou em outras palavras, acredita que podemos transformar *qualquer* algoritmo probabilístico polinomial em um algoritmo determinístico também polinomial (ARORA; BARAK, 2009).

A classe de complexidade **BPP** captura o que chamamos de *erro nos dois lados*, pois permite (com baixa probabilidade) a uma MTP M, que decide a linguagem $L \subseteq \{0,1\}^*$, ao rodar com a string x responder ambos 0 se $x \in L$ e 1 se $x \notin L$. Alguns algoritmos, porém, possuem o que chamamos de *erro em um lado*: nunca respondem 1 se $x \notin L$, apesar de poderem responder 0 se $x \in L$, por exemplo. Esse tipo de comportamento é capturado pela classe **RP**, definida a seguir:

Definição 4.5. (As classes **RTIME** e **RP**) (GILL, 1974) **RTIME**$(T(n))$ contém todas as linguagens $L \subseteq \{0,1\}^*$ para as quais existem uma MTP M de tempo $T(n)$ tal que

$$x \in L \to \Pr[M(x) = 1] \geq \frac{2}{3}$$

$$x \notin L \to \Pr[M(x) = 1] = 0$$

Então, $\mathbf{RP} = \bigcup_{c>0} \mathbf{RTIME}(n^c)$

Em especial, $\mathbf{RP} \subseteq \mathbf{NP}$ pois qualquer caminho que aceite a string é um certificado de que essa pertence a linguagem. Por outro lado, não sabemos se $\mathbf{BPP} \subseteq \mathbf{NP}$ (ARORA; BARAK, 2009). Definimos também a classe $\mathbf{coRP} = \{L; \overline{L} \in \mathbf{RP}\}$, que captura erro no outro lado (pode responder 1 se $x \notin L$ mas nunca responde 0 se $x \in L$).

Uma outra classe de complexidade, que envolve as MTPs que nunca erram suas respostas, porém rodam em tempo *esperado* polinomial, é a classe **ZPP**, definida a seguir:

Definição 4.6. (Tempo esperado de uma MTP) Para uma MTP M e uma string x, definimos a variável aleatória $T_{M,x}$ como o tempo de execução de M com x. Isso é, $\Pr[T_{M,x} = T] = p$ se com probabilidade p sobre as escolhas aleatórias de M com a entrada x, M para em até T passos. Dizemos que M possui *tempo esperado de execução* $T(n)$ se a esperança $\mathrm{E}[T_{M,x}]$ é no máximo $T(|x|)$ para todo $x \in \{0,1\}^*$.

Definição 4.7. (As classes **ZTIME** e **ZPP**) (GILL, 1974) A classe $\mathbf{ZTIME}(T(n))$ contém todas as linguagens $L \subseteq \{0,1\}^*$ para as quais existem uma MTP M que roda em tempo esperado $O(T(n))$ tal que, para qualquer entrada $x \in \{0,1\}^*$, $M(x) = L(x)$. Então $\mathbf{ZPP} = \bigcup_{c>0} \mathbf{ZTIME}(T(n))$.

O seguinte teorema mostra a relação não muito intuitiva entre as classes **ZPP**, **RP** e **coRP**. É comum que a prova desse teorema seja omitida ou então deixada como exercício em livros de Complexidade Computacional (ARORA; BARAK, 2009; PAPADIMITRIOU, 1994).

Teorema 4.2. *(Relação entre **ZPP**, **RP** e **coRP**)* $\boldsymbol{ZPP = RP \cap coRP}$.

Demonstração. Provar que $\mathbf{ZPP} = \mathbf{RP} \cap \mathbf{coRP}$ é mais simples quando utilizamos uma definição alternativa para a classe **ZPP**:

Afirmação 4.2. *Uma linguagem $L \in \boldsymbol{ZPP}$ se e somente se existe uma MTP M' de tempo polinomial*[1] *que responde um dentre $\{0, 1, \text{NS}\}$, tal que para toda entrada $x \in \{0,1\}^*$, com probabilidade 1, $M'(x) \in \{L(x), \text{NS}\}$ e $Pr[M(x) = \text{NS}] \leq 1/2$. Isso é, se M' responde 1 ou 0, essa resposta está certa, porém ainda pode responder* NS *("não sei") até metade das vezes.*

Demonstração. Se $L \in \mathbf{ZPP}$, então existe uma MTP M de tempo esperado $O(n^c)$, tal que, para toda entrada $x \in \{0,1\}^*$, quando M para rodando com x o resultado $M(x)$ é

[1] Note que aqui a máquina M' roda em tempo polinomial e não em tempo *esperado* polinomial.

igual a $L(x)$. Podemos construir M' através de M da seguinte maneira: para qualquer $x \in \{0,1\}^*$, tal que $|x| = n$, basta executar $M(x)$ com um contador, limitando sua execução à $2Cn^c$ passos, se M rodando com x para dentro desse limite de tempo, então $M'(x) = M(x) = L(x)$, senão, $M'(x) =$ NS. Dessa forma, $M'(x)$ só responde NS se $M(x)$ executar mais que $2Cn^c$ passos, ao denotar por X a variável aleatória que corresponde ao tempo de execução de M e levando em consideração que $\mathrm{E}[X] \leq Cn^c$, temos, pela desigualdade de Markov:

$$\Pr[M'(x) = \text{NS}] = \Pr[X > 2Cn^c] \leq \frac{1}{2}$$

Também podemos construir M a partir de M' da seguinte maneira: para qualquer $x \in \{0,1\}^*$, tal que $|x| = n$, basta executar $M'(x)$. Se M' rodando com x responder 1 ou 0, então a computação para e $M(x) = M'(x) = L(x)$, já se $M'(x)$ responder NS, executa-se novamente M' com a entrada x, quantas vezes for necessário até que essa responda 1 ou 0. Dessa forma serão executados até Cn^c passos por execução de M' com a entrada x, e como $\Pr[M'(x) = \text{NS}] \leq 1/2$, denotando por Y a variável aleatória que corresponde a quantas execuções de M' com a entrada x são necessárias para obter um resultado 0 ou 1, ou então um "sucesso" (variável aleatória geométrica), e por X a variável aleatória que corresponde ao tempo de execução de M, temos que:

$$\mathrm{E}[X] \leq \mathrm{E}[Cn^c \cdot Y] = Cn^c \cdot \mathrm{E}[Y] = 2Cn^c = O(n^c)$$

□ (afirmação)

Agora que temos essa definição alternativa de **ZPP**, provamos primeiro que **RP** ∩ **coRP** ⊆ **ZPP**. Se $L \in \mathbf{RP} \cap \mathbf{coRP}$, então existem duas MTP's M_{RP} e M_{coRP} tal que para qualquer entrada $x \in \{0,1\}^*$:

$$\text{Se } L(x) = 0, \text{ então } M_{RP}(x) = 0 \text{ e } \Pr[M_{coRP}(x) = 0] \geq \frac{2}{3}$$

$$\text{Se } L(x) = 1, \text{ então } M_{coRP}(x) = 1 \text{ e } \Pr[M_{RP}(x) = 1] \geq \frac{2}{3}$$

Podemos então construir M' da seguinte maneira: para qualquer $x \in \{0,1\}^*$, execute $M_{RP}(x)$ e $M_{coRP}(x)$, e então

$$\text{Se } M_{RP}(x) = 1, \text{ então } M'(x) = 1$$

$$\text{Se } M_{coRP}(x) = 0, \text{ então } M'(x) = 0$$

$$\text{Se } M_{RP}(x) = 0 \text{ e } M_{coRP}(x) = 1, \text{ então } M'(x) = \text{NS}$$

Basta então verificar que $\Pr[M'(x) = \text{NS}] \leq 1/2$:

$$\text{Se } L(x) = 1, \text{ então } \Pr[M_{RP}(x) = 0] \leq \frac{1}{3} \text{ e } \Pr[M_{coRP}(x) = 1] \geq \frac{2}{3} \leq 1$$

$$\text{Se } L(x) = 0, \text{ então } \Pr[M_{coRP}(x) = 0] \leq \frac{1}{3} \text{ e } \Pr[M_{RP}(x) = 1] \geq \frac{2}{3} \leq 1$$

Dessa forma:

$$\Pr[M'(x) = \text{NS}] = \Pr[M_{RP}(x) = 0 \cap M_{coRP}(x) = 1] \leq \frac{1}{3}$$

Agora mostramos que **ZPP** $\subseteq$ **RP**. Se $L \in$ **ZPP**, então existe a MTP M' discutida anteriormente. Podemos então construir M_{RP} da seguinte maneira: para qualquer $x \in \{0,1\}^*$, execute $M'(x)$ duas vezes, e então

$$\text{Se em alguma das execuções, } M'(x) = 1, \text{ então } M_{RP}(x) = 1$$

$$\text{Se em alguma das execuções, } M'(x) = 0, \text{ então } M_{RP}(x) = 0$$

$$\text{Se em ambas as execuções, } M'(x) = \text{NS}, \text{ então } M_{RP}(x) = 0$$

Construindo M_{RP} dessa forma, garantimos que essa nunca erra quando a string x não pertence a L. Definimos X como a variável aleatória que indica quantas vezes (uma ou duas) M' rodando com x responde NS. O comportamento da máquina M_{RP} construída é o seguinte:

$$\text{Se } L(x) = 0, \text{ então } M_{RP}(x) = 0$$

$$\text{Se } L(x) = 1, \text{ então } \Pr[M_{RP}(x) = 1] = \Pr[X \neq 2] \geq 1 - \frac{1}{4} \geq \frac{2}{3}$$

Ou seja, $L \in$ **RP**. Podemos mostrar que **ZPP** $\subseteq$ **coRP** construindo M_{coRP} a partir de M' de forma totalmente análoga, com a diferença de que M_{coRP} responde 1 ao rodar com x caso ambas as execuções de $M'(x)$ resultarem na saída NS. □

Já os seguintes teoremas indicam a relação da classe **BPP** com a classe $\mathbf{P}_{/\text{poly}}$ e com a hierarquia polinomial:

Teorema 4.3. *(Relação entre **BPP** e $\boldsymbol{P}_{/poly}$) (ADLEMAN, 1978)* $\boldsymbol{BPP} \subseteq \boldsymbol{P}_{/poly}$

Demonstração. Suponha que $L \in$ **BPP**, então pela definição alternativa de **BPP** e pelo Teorema 4.1 existe uma MT M que em entradas de tamanho n usa m bits aleatórios tal que para todo $x \in \{0,1\}^n$, $\Pr_r[M(x,r) \neq L(x)] \leq 2^{-n-1}$. Dizemos que uma string $r \in \{0,1\}^m$ é "ruim" para $x \in \{0,1\}^n$ se $M(x,r) \neq L(x)$ e caso contrário r é "boa" para x. Para todo $x \in \{0,1\}^n$, no máximo $2^m/2^{n+1}$ strings r são ruins. Somando para todos os

$x \in \{0,1\}^n$, existem no máximo $2^n \cdot (2^m/2^{n+1}) = 2^m/2$ strings que são ruins para algum x. Em particular, existe uma string $r_0 \in \{0,1\}^m$ que é boa para todo x. Podemos usar essa string r_0 como um conselho para a MT M para as strings de tamanho n, dessa forma $M(x, r_0) = L(x)$ para todo $x \in \{0,1\}^n$. Pela caracterização alternativa de $\mathbf{P_{/poly}}$ do Teorema 2.8, temos que $L \in \mathbf{P_{/poly}}$. □

Teorema 4.4. *(Teorema Sipser-Gacs-Lautemann) (SIPSER, 1983; LAUTEMANN, 1983)* $\boldsymbol{BPP} \subseteq \boldsymbol{\Sigma}_2^p \cap \boldsymbol{\Pi}_2^p$

4.3 PROVAS INTERATIVAS

Quando refletimos sobre o conceito de uma prova matemática, nos deparamos com um conceito semelhante à definição da classe **NP**. Para convencer alguém de que determinada afirmação é verdadeira, escrevemos uma sequência de símbolos em um papel (certificado), que é então verificada por quem queremos convencer. Percebe-se então que há uma **interação**, ainda que simples, entre duas entidades: um *provador* e um *verificador*.

O estudo de provas interativas foca na complexidade da interação entre essas duas entidades, com algumas restrições (GOLDWASSER et al., 1989). Primeiramente, é necessário que seja possível provar uma afirmação verdadeira, e que seja impossível (ou que haja uma probabilidade muito baixa de) provar uma afirmação falsa. Além disso, é importante que a interação seja eficiente, no sentido de que não interessa o poder computacional que o provador tem a sua disposição, contanto que na interação a computação necessária por parte do verificador seja eficiente (polinomial).

O tipo de interação mais simples que se pode ter é a interação entre duas entidades (representadas por funções) determinísticas:

Definição 4.8. (Interação entre funções determinísticas) Sejam $f, g : \{0,1\}^* \to \{0,1\}^*$ funções e $k \geq 0$ um número inteiro (que pode depender do tamanho da entrada). Uma *interação de k rodadas* entre f e g com a entrada $x \in \{0,1\}^*$, denotada por $\langle f, g \rangle (x)$ é uma sequência de strings $a_1, \ldots, a_k$ definida da seguinte maneira:

$$a_1 = f(x)$$

$$a_2 = g(x, a_1)$$

$$\ldots$$

$$a_{2i+1} = f(x, a_1, \ldots, a_{2i}) \text{ para } 2i < k$$

$$a_{2i+2} = g(x, a_1, \ldots, a_{2i+1}) \text{ para } 2i + 1 < k$$

A saída de f no fim da interação, que é denotada por $\text{OUT}_f \langle f, g \rangle (x)$, é definida como o resultado de $f(x, a_1, \ldots, a_k)$ e como sendo um elemento do conjunto $\{0, 1\}$.

Tendo definido a interação entre duas funções, podemos agora definir um *sistema de prova determinístico*, em que uma das funções (o provador) apresenta mensagens para convencer a outra (o verificador) de que a string de entrada x pertence a determinada linguagem L:

Definição 4.9. (Sistema de prova determinístico) Dizemos que uma linguagem $L \subseteq \{0, 1\}^*$ possui um *sistema interativo determinístico de prova de k rodadas* se existe uma MT V que, na entrada $x, a_1, \ldots, a_i$, roda em tempo polinomial em $|x|$ e possui uma interação de k rodadas com qualquer função P tal que:

$$\text{(Integralidade) } x \in L \rightarrow \exists P : \{0,1\}^* \rightarrow \{0,1\}^*; \text{OUT}_V \langle V, P \rangle (x) = 1$$

$$\text{(Consistência) } x \notin L \rightarrow \forall P : \{0,1\}^* \rightarrow \{0,1\}^*; \text{OUT}_V \langle V, P \rangle (x) = 0$$

Definimos também a classe **dIP**, que contém todas as linguagens para as quais existe um sistema de prova determinístico que possua no máxima uma quantidade polinomial (no tamanho da entrada) de rodadas. Porém essa classe se mostra não muito interessante, devido ao seguinte teorema:

Teorema 4.5. *(ARORA; BARAK, 2009)* ***dIP*** $=$ ***NP***

Demonstração. É fácil perceber que toda linguagem L em **NP** possui um sistema de prova determinístico, basta que o provador envie o certificado da string $x \in L$ e que o verificador execute o algoritmo verificador polinomial para se certificar de que a string x realmente pertence a L. Por outro lado, caso $L \in$ **dIP**, mostramos que $L \in$ **NP**: Seja V o verificador polinomial do sistema de prova determinístico, um certificado para a string $x \in L$ é a sequência de mensagens $(a_1, \ldots, a_k)$ que fazem com que V aceite x, e basta verificar que $V(x) = a_1$, $V(x, a_1, a_2) = a_3$ e assim por diante até $V(x, a_1, \ldots, a_k) = 1$. Como definimos um certificado e verificador polinomial para L, $L \in$ **NP**. □

Para que haja um salto no poder de provas interativas, é necessário permitir que o verificador seja *probabilístico* (e que, por esse motivo, o verificador tenha uma pequena probabilidade de aceitar uma afirmação falsa) (GOLDWASSER et al., 1989).

Estendemos então a definição de interação para que permita ao verificador se utilizar de *moedas privadas*, que são bits aleatórios visíveis somente para o verificador. Para que f seja probabilística, adicionamos uma entrada r de bits aleatórios à função f, dessa forma $a_1 = f(x, r)$, $a_3 = f(x, r, a_1, a_2)$ e assim por diante. Porém a entrada r não é comunicada à função g, que continua computando suas respostas apenas nas entradas x e nas mensagens a_i que f envia. A interação entre f e g se torna então uma variável aleatória sobre $r \in \{0,1\}^m$. Da mesma forma, o resultado da interação $\text{OUT}_V \langle V, P \rangle (x)$ também é uma variável aleatória.

Definimos então a classe **IP**:

Definição 4.10. (Verificadores probabilísticos e a classe **IP**) Para um inteiro $K \geq 1$, que pode depender do tamanho da entrada, dizemos que a linguagem L está em $\mathbf{IP}[k]$ se existe uma MTP de tempo polinomial V que realiza uma interação de k rodadas com um provador $P : \{0,1\}^* \to \{0,1\}^*$, satisfazendo as seguintes condições:

$$\text{(Integralidade) } x \in L \to \exists P : \{0,1\}^* \to \{0,1\}^*; \Pr[\text{OUT}_V \langle V, P \rangle (x) = 1] \geq \frac{2}{3}$$

$$\text{(Consistência) } x \notin L \to \forall P : \{0,1\}^* \to \{0,1\}^*; \Pr[\text{OUT}_V \langle V, P \rangle (x) = 1] \leq \frac{1}{3}$$

onde todas as probabilidades são em função de r (a sequência de bits aleatórios de V). Definimos $\mathbf{IP} = \bigcup_{c \geq 1} \mathbf{IP}[n^c]$.

Assim como no caso das classes de complexidade **BPP**, **RP** e **coRP**, podemos mudar os valores $2/3$ e $1/3$ por qualquer valor acima e abaixo de $1/2$, respectivamente, sem mudar a classe **IP** (inclusive, por um teorema não trivial, podemos trocar a probabilidade da condição de Integralidade por 1, também sem alterar a classe **IP**). E ainda podemos repetir o protocolo várias vezes para garantir que a probabilidade de que uma string que não pertença a L seja aceita ou de que uma string que pertença a L não seja aceita sejam muito baixas (ARORA; BARAK, 2009).

O seguinte teorema mostra que quando permitimos ao verificador utilizar-se de aleatoriedade, há um grande salto (sob conjecturas fortes da área de Complexidade Computacional) no tipo de linguagem que possui protocolos interativos:

Teorema 4.6. *(LUND et al., 1992; SHAMIR, 1992)* ***IP = PSPACE****.*

Para demonstrar o poder desse tipo de interação, mostramos um protocolo, retirado de (ARORA; BARAK, 2009), para a linguagem GNI, ou o problema de não isomorfismo de grafos (um problema da classe **coNP** e cuja pertinência à classe **NP** continua

em aberto):

Protocolo para GNI - Ambas as partes conhecem os grafos G_0 e G_1, de n vértices V: Escolhe b dentre $\{0,1\}$ uniformemente, então sorteia uma permutação $\pi \in_R S_n$ e aplica ao grafo G_b, obtendo um novo grafo H, que é enviado para P. P: Identifica qual dos grafos G_0 ou G_1 foi utilizado para obter H. Seja G_i esse grafo, envia i para V. V: Aceita se $i = b$, rejeita caso contrário.

Percebe-se que caso os grafos não sejam isomorfos existe P que consegue distinguir entre os dois e enviar a resposta correta para V, fazendo com que V aceite com probabilidade 1. Por outro lado, caso G_0 e G_1 sejam isomorfos, não há como saber qual dos dois foi utilizado para obter H, e qualquer P pode no máximo tentar adivinhar (com probabilidade $1/2$) a resposta.

4.4 PROVAS DE CONHECIMENTO ZERO

Em um sistema de prova interativa, é possível que o verificador V, ao fim da interação com P, obtenha mais informações além da informação de que a afirmação é verdadeira. Esse é o caso dos protocolos de prova determinística para problemas da classe **NP**, nos quais o provador envia o certificado de pertinência da string para o verificador, que além de se convencer de que a string pertence a linguagem em questão, agora também possui conhecimento do certificado para esse fato.

A ideia das provas interativas de conhecimento zero é a de que no fim da interação o verificador apenas obtém a informação de que a afirmação é verdadeira, e nada além disso (GOLDWASSER et al., 1989). Problemas que possuem provas interativas de conhecimento zero possuem conexões importantes com o problema de minimização de circuitos (ALLENDER; DAS, 2014), o que torna o estudo desse tipo de prova interativa importante para os objetivos desse trabalho.

Dessa forma, protocolos de prova interativa de conhecimento zero possuem, além das condições de *consistência* e *integralidade* vistas anteriormente, a condição de *conhecimento zero*, que indica que, caso a afirmação seja verdadeira, qualquer conhecimento que o verificador tenha adquirido durante a interação poderia ter sido computado por ele mesmo, sem necessidade de qualquer tipo de interação. A seguinte definição formaliza esse conceito para linguagens em **NP**:

Definição 4.11. (Protocolos de Conhecimento Zero) Seja $L \in \mathbf{NP}$ uma linguagem e M o verificador para L, ou seja, $x \in L \leftrightarrow \exists u \in \{0,1\}^{p(|x|)}; M(x,u) = 1$, onde p é um polinômio. Um par P e V de MTPs de tempo polinomial é chamado de protocolo interativo de conhecimento zero para L caso essas três condições sejam cumpridas:

(Integralidade) Para todo $x \in L$ e u um certificado para esse fato, (ou seja, $M(x,u) = 1$), $\Pr[\text{OUT}_V \langle P(x,u), V(x)\rangle = 1] \geq 2/3$, onde $\langle P(x,u), V(x)\rangle$ denota a interação entre V com a entrada x e P com as entradas x e u (x e seu certificado de pertinência a L), e $\text{OUT}_V I$ denota a saída de V no fim da interação I.

(Consistência) Se $x \notin L$, então para qualquer P^* e entrada $u \in \{0,1\}^{p(|x|)}$, $\Pr[\text{OUT}_V \langle P(x,u), V(x)\rangle = 1] \leq 1/3$ (note que P^* não precisa rodar em tempo polinomial)

(Conhecimento Zero Perfeito) Para toda MTP polinomial V^*, existe uma MTP de tempo esperado polinomial S^* (o simulador de V^*) tal que para todo $x \in L$ e u um certificado para esse fato, $\text{OUT}^*_V \langle P(x,u), V^*(x)\rangle \equiv S^*$, isso é, essas duas variáveis aleatórias são identicamente distribuídas mesmo que S^* não possua um certificado para a instância x.

Na definição anterior foi apresentado o conceito de provas de conhecimento zero perfeito, porém também existem provas de conhecimento zero estatístico, em que basta que as distribuições do simulador S^* e da interação do verificador V^* com P possuam distância estatística pequena, e provas de conhecimento zero computacional, nas quais as duas distribuições devem ser computacionalmente indistinguíveis. Definimos então as classes **PZK**, **SZK** e **CZK**, que contém as linguagens que possuem protocolos de conhecimento zero dos três tipos apresentados, respectivamente. É importante notar que conjectura-se que **SZK** está estritamente contida entre **P** e **NP** (ARORA; BARAK, 2009), o que a coloca como uma classe importante para esse trabalho.

A seguir, apresentamos um protocolo de conhecimento zero perfeito para o problema de Isomorfismo de Grafos (perceba que como GI $\in$ **NP**, há um sistema de prova determinístico trivial para a linguagem, porém esse sistema revela o certificado para a instância verdadeira):

Protocolo para GI - Ambas as partes conhecem os grafos G_0 e G_1 de n vértices, P conhece também uma permutação π que mapeia G_0 em G_1, ou seja, $\pi(G_0) = G_1$.
P: Sorteia uma permutação aleatória $\pi_1 \in_R S_n$ e aplica essa permutação ao grafo G_0, obtendo $H = \pi_1(G_0)$, que é enviado para V.
V: Escolhe b dentre $\{0, 1\}$ uniformemente e então envia o bit b para P.
P: Se $b = 0$, envia a permutação π_1 para V. Caso $b = 1$, envia a composição de π e π_1, denominada $\pi \circ \pi_1$, para V.
V: Seja π_2 a permutação recebida na última mensagem. Se $b = 0$, verifica se $\pi_2(G_0) = H$. Caso $b = 1$, verifica se $\pi_2(G_1) = H$. Aceita se a verificação obtiver sucesso e rejeita caso contrário.

Teorema 4.7. *(GOLDREICH et al., 1991) (GI ∈ **PZK**)*

Demonstração. A prova envolve provar as três condições que caracterizam um protocolo de conhecimento zero perfeito.

(Integralidade) Percebe-se claramente nesse protocolo que se ambos P e V seguirem o protocolo e $G_0 \cong G_1$, então a probabilidade de que P aceita é 1.

(Consistência) Caso $G_0 \ncong G_1$, independente da estratégia que P^* utilize para construir o grafo H, este só poderá ser isomorfo ou a G_0 ou a G_1. Dessa forma, com probabilidade $1/2$ V irá escolher $b \in_R \{0, 1\}$ de forma que $H \ncong G_b$. Como nesse caso não existe permutação que mapeie H em G_b, o verificador irá rejeitar. a repetição do protocolo coloca essa probabilidade abaixo de $1/3$.

(Conhecimento Zero Perfeito) Para provar essa condição é necessário apresentar um simulador S^* para a interação entre qualquer MTP polinomial V^* e a máquina P nas strings $x \in L$ (instâncias de GI em que $G_0 \cong G_1$), além disso é preciso mostrar que S^* roda em tempo esperado polinomial e que a distribuição das mensagens que envia para V^* é igual a distribuição das mensagens que P envia. Apresentamos então o simulador S^*:

Na entrada (G_0, G_1), onde ambos os grafos possuem n vértices, sorteia um bit aleatório $b' \in_R \{0, 1\}$ e uma permutação π aleatória do conjunto de permutações de n vértices, e então computa $H = \pi(G_{b'})$. H é então enviado para V^*, que responde com um bit (que pode não ser aleatório) $b \in \{0, 1\}$. Se $b = b'$, S^* envia π para V^* e tem como saída a saída de V^*. Por outro lado (se $b \neq b'$), S^* reinicia a simulação.

Uma observação crucial é a de que a primeira mensagem de S^* é distribuída identicamente à primeira mensagem de P, pois ambos enviam um grafo aleatório H que

é isomorfo tanto a G_0 quanto a G_1 (lembrando que a condição de conhecimento zero só se aplica para instâncias verdadeiras). Isso significa que o grafo H não revela qual o valor de b', dessa forma V^* não pode escolher b com base nessa informação e a probabilidade de que $b = b'$ é $1/2$. No caso de $b = b'$, a segunda mensagem de S^* também é identicamente distribuída à segunda mensagem de P^* (uma permutação aleatória π que mapeia ou G_0 ou G_1 a H). Por último, como $\Pr[b = b'] = 2$, a probabilidade de que sejam necessárias k interações para que isso ocorra é 2^{-k}, portanto o tempo esperado de execução de S^* é $T(n)\sum_{k=1}^{\infty} 2^{-k} = O(T(n))$, onde $T(n)$ é o tempo de execução de V^*, como esse tempo é polinomial o tempo esperado de S^* também é polinomial.

□

Em especial, o protocolo apresentado para GI é do tipo "v-bit" (*verifier bit*), pois segue a seguinte sequência de mensagens: P envia uma mensagem, V envia um bit aleatório, P responde e V faz uma verificação para aceitar ou não. A maior parte dos problemas para os quais foram mostrados protocolos de conhecimento zero perfeito admitem protocolos com essas características (MALKA, 2008).

Assim como várias outras classes apresentadas, as classes **PZK** e **SZK** possuem problemas completos, porém os problemas dessas classes são melhor caracterizados como *problemas de promessa*, definidos a seguir:

Definição 4.12. (ALLENDER; DAS, 2014) (Problema de promessa) Um *problema de promessa* P é um par de linguagens disjuntas (S, N), onde o conjunto S representa as instâncias "Sim" de P e N representa as instâncias "Não" de P.

A ideia de problemas de promessa é que eles restringem o conjunto de strings para as quais o problema é importante, dessa forma podemos, por exemplo, para o problema SAT, assumir que a entrada é uma fórmula booleana (pois nem todas as strings são representações válidas de fórmulas booleanas).

Apresentamos os problemas distância estatística ($\mathrm{SD}^{0,1}$) e densidade de intersecção de imagem polarizada (PIID), que são completos para as linguagens de **PZK** com protocolos do tipo "v-bit" e **SZK**, respectivamente (MALKA, 2008; BEN-OR; GUTFREUND, 2003).

No problema $\mathrm{SD}^{0,1}$, temos como entrada dois circuitos (D_0, D_1) de tamanho n e com entradas de tamanho m. As instâncias "Sim" de $\mathrm{SD}^{0,1}$ são aquelas em que

$\Delta(D_0, D_1) = 0$, ou seja, os dois circuitos representam distribuições idênticas[2]. Já as instâncias "Não" são pares (D_0, D_1) onde os circuitos possuem imagens disjuntas, ou então distância estatística 1.

O problema PIID é semelhante, as entradas para o problema são triplas do tipo (n, D_0, D_1), onde D_0 e D_1 são circuitos de tamanho no máximo n^k (para algum k fixo) com entradas de tamanho m. As instâncias "Sim" de PIID são as triplas (n, D_0, D_1) onde $\Delta(D_0, D_1) \leq 1/2^n$, ou seja, a distância estatística entre as distribuições que os circuitos representam é no máximo $1/2^n$. Já as instâncias "Não" são triplas (n, D_0, D_1) com a propriedade de que $\Pr_{|x|=m}[\exists y; D_1(y) = D_0(x)] < 1/2_n$.

4.5 GERADORES PSEUDOALEATÓRIOS E FUNÇÕES UNIDIRECIONAIS

Outra aplicação muito importante dos conceitos de aleatoriedade e complexidade computacional se dá na área de Criptografia. Nessa área, a tarefa fundamental é a de *cifragem*, na qual uma mensagem, no formato de uma sequência de bits, é transformada em outra, ilegível, que pode apenas ser resgatada pelo destinatário da mensagem. Alguns conceitos centrais para a área de Criptografia e com fortes conexões entre si são geradores pseudoaleatórios e funções direcionais. Esses conceitos são importantes para os objetivos desse trabalho pois alguns dos problemas candidatos a **NP**-intermediários estudados (como resíduo quadrático e logaritmo discreto) envolvem a inversão de funções candidatas a unidirecionais.

A ideia básica de cifragem, através do conceito de *criptografia simétrica*, é ilustrada na Figura a seguir:

Ambos Alice e Bob possuem uma chave aleatória previamente compartilhada $k \in_R \{0,1\}^n$. Para enviar uma mensagem $x \in \{0,1\}^m$ para Bob, Alice produz $y = E_k(x)$, onde E é o algoritmo de cifragem que produz um texto cifrado através de x e k. Para recuperar a mensagem, Bob deve executar o algoritmo D que decifra a mensagem y, o que é denotado por $D_k(y) = x$. Caso Eva intercepte a mensagem y, espera-se que ela não possa obter nenhuma informação em relação à x.

Certamente esses conceitos necessitam de formalização, por exemplo, o que significa dizer que Eva não pode obter nenhuma informação do texto cifrado? Shannon (1949a) propõe o conceito de *sigilo perfeito*, que formaliza essa noção:

[2]Note que podemos assumir que circuitos representam distribuições estatísticas, por exemplo, se existem i entradas que fazem D_0 responder x, então $\Pr[D_0 = x] = i/2^m$.

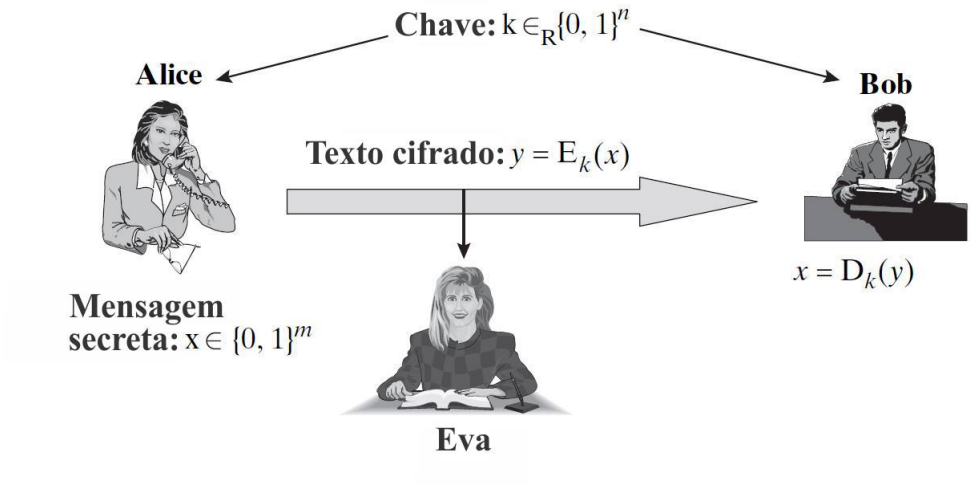

Figura 6: Esquema de criptografia simétrica.
Fonte: Traduzido de (ARORA; BARAK, 2009).

Definição 4.13. (Sigilo perfeito) Seja (E, D) um esquema de criptografia para mensagens de tamanho m com chave de tamanho n que satisfaça $D_k(E_k(x)) = x$. Dizemos que (E, D) possui *sigilo perfeito* se para todo par de mensagens $x, x' \in \{0,1\}^m$, as distribuições de $E_{U_n}(x)$ e $E_{U_n}(x')$ são idênticas.

A ideia então é que Eva sempre enxerga a mesma distribuição de mensagens, independente da mensagem que é cifrada, dessa maneira Eva não consegue obter nenhuma informação sobre a mensagem original.

Uma maneira simples, porém com várias limitações, de implementar um esquema de criptografia com sigilo perfeito é a *cifra de uso único* (*one time pad* em inglês), que funciona da seguinte maneira: Para cifrar uma mensagem $x \in \{0,1\}^n$, escolhemos aleatoriamente uma chave de mesmo tamanho $k \in_R \{0,1\}^n$ e ciframos x fazendo o XOR bit a bit de x com k, ou então $x \oplus k$. O destinatário então realiza novamente o XOR bit a bit de k com $x \oplus k$ e recupera a mensagem. Para verificar o sigilo perfeito desse esquema basta observar que a mensagem $x \oplus k$ é identicamente distribuída a qualquer outra $x' \oplus k$ pois a probabilidade de que $x \oplus k = z$ é a mesma de que $k = x \oplus z$, e como k é uma string aleatória, essa probabilidade é de $1/2^n$.

Percebe-se que o esquema apresentado possui duas limitações importantes, a primeira é que a chave k não pode ser reutilizada, pois caso contrário um adversário poderia realizar a operação $(x \oplus k) \oplus (x' \oplus k)$ e obter $(x \oplus x')$, que é uma informação não trivial sobre as mensagens. Além disso, é necessária uma chave do tamanho da mensagem que

será enviada, o que inviabiliza esse tipo de esquema para mensagens muito grandes.

Para tentar mitigar essas limitações, surge o conceito de *geradores pseudoaleatórios*, que possuem o objetivo de "esticar" chaves de tamanho n (denominadas sementes) para outras muito maiores de tamanho m que ainda são "aleatórias" o bastante para qualquer algoritmo polinomial, podendo ser usadas com a cifra de uso único. Antes de definir esse conceito, porém, apresentamos o de funções negligíveis, que será útil nesta seção:

Definição 4.14. (Funções negligíveis) Uma função $\epsilon : \mathbb{N} \to [0, 1]$ é chamada negligível se $\epsilon(n) = n^{-\omega(1)}$.

A ideia é que funções negligíveis tendem a zero rapidamente, portanto eventos que ocorrem com probabilidade negligível podem ser ignorados sem problemas na maior parte dos cenários (ARORA; BARAK, 2009). Podemos então definir geradores pseudoaleatórios:

Definição 4.15. (Geradores pseudoaleatórios seguros) Sejam $G : \{0,1\}^* \to \{0,1\}^*$ e $\ell : \mathbb{N} \to \mathbb{N}$ funções computáveis em tempo polinomial tal que $\ell(n) > n$ para todo n. Dizemos que G é um *gerador pseudoaleatório seguro com esticamento* $\ell(n)$, se $|G(x)| = \ell(|x|)$ para todo $x \in \{0,1\}^*$ e se para todo algoritmo probabilístico polinomial A, existe uma função negligível $\epsilon : \mathbb{N} \to [0, 1]$ tal que:

$$|\Pr[A(G(U_n)) = 1] - \Pr[A(U_{\ell(n)}) = 1]| < \epsilon(n)$$

para todo $n \in \mathbb{N}$.

Essa definição indica que para um gerador pseudoaleatório G ser seguro, qualquer algoritmo probabilístico polinomial deve ter probabilidade negligível de distinguir uma string de tamanho $\ell(n)$ gerada por G de uma string de mesmo tamanho que é realmente aleatória. Essa condição é suficiente para que esse tipo de gerador pseudoaleatório seja utilizado para gerar chaves que serão utilizadas com a cifra de uso único.

Outro conceito que é muito importante na área de criptografia são as *funções unidirecionais*, que de forma simplificada são funções fáceis de computar porém difíceis de inverter, esse conceito é formalizado a seguir:

Definição 4.16. (Funções unidirecionais) uma função computável em tempo polinomial $f : \{0,1\}^* \to \{0,1\}^*$ é uma *função unidirecional* se para todo algoritmo probabilístico

polinomial A, existe uma função negligível $\epsilon : \mathbb{N} \to [0, 1]$ tal que:

$$\Pr_{\substack{x \in_R \{0,1\}^n \\ y = f(x)}} [A(y) = x'; f(x') = y] < \epsilon(n)$$

para todo $n \in \mathbb{N}$.

Há muitas funções candidatas a serem funções unidirecionais, como multiplicação (cujo inverso é fatoração), exponenciação módulo um inteiro n (cujo inverso é o logaritmo discreto), entre outras. Porém provar que essas funções realmente possuem essa propriedade é extremamente difícil, pois esse é um resultado que implicaria $\mathbf{P} \neq \mathbf{NP}$ (ARORA; BARAK, 2009).

Percebe-se ainda semelhanças nas definições dos conceitos de geradores pseudoaleatórios e funções unidirecionais. Um dos motivos para esse fato é que os dois conceitos são equivalentes, isso é, podemos construir qualquer um dos dois a partir do outro, com *overhead* no máximo polinomial. O lado não trivial dessa igualdade (construir um gerador pseudoaleatório a partir de uma função unidirecional) foi provado por Hästad et al. (1999) e é considerado um dos resultados mais importantes da área de complexidade computacional.

5 PROBLEMAS CANDIDATOS A NP-INTERMEDIÁRIOS

Neste capítulo apresentamos quatro problemas candidatos a **NP**-intermediários, os problemas de *minimização de circuitos*, *isomorfismo de grafos*, *resíduo quadrático* e *logaritmo discreto*. Além disso são apresentados argumentos para justificar essas conjecturas.

5.1 MINIMIZAÇÃO DE CIRCUITOS

O problema de minimização de circuitos, ou MCSP (em inglês *Minimum Circuit Size Problem*) é um problema candidato a ser **NP**-intermediário (ALLENDER; DAS, 2014). A definição do problema é a seguinte: dadas uma função booleana f em n variáveis através da sua tabela verdade de tamanho 2^n e um número $k \in \mathbb{N}^*$, queremos saber se existe um circuito Booleano de tamanho k que computa f. Sua definição através de um problema de decisão é a seguinte:

$$\text{MCSP} = \{\langle f, k\rangle \,;\ \text{existe um circuito Booleano de tamanho } k \text{ que computa } f\}$$

Nessa definição o tamanho do circuito é a quantidade de conectivos que esse possui. Apesar de essa não ser a mesma utilizada para definir as classes $\mathbf{SIZE}(T(n))$ de medida de tamanho de famílias de circuitos, percebe-se que ambas são polinomialmente equivalentes, pois um circuito C com n *gates* pode ter no máximo n^2 conectivos.

Pouco se sabe sobre a complexidade de MCSP. Se por um lado acredita-se que o problema não esteja em **P** (o capítulo 6 provê evidências nesta direção), por outro, não é possível provar (MURRAY; WILLIAMS, 2014) a **NP**-completude do problema através de reduções por "gadgets" (como é o caso de várias reduções clássicas de **NP**-completude (KARP, 1972)). Além disso, uma prova de **NP**-completude utilizando reduções independentes a oráculo (todas as reduções para o problema até o momento possuem essa característica) causaria um colapso ao segundo nível na hierarquia polinomial (HIRAHARA; WATANABE, 2015).

Ainda assim existe a possibilidade de o problema ser **NP**-completo, porém uma prova para essa afirmação (através de redução polinomial) implica que **ZPP** $\neq$ **EXP** (MURRAY; WILLIAMS, 2014). Apesar de a maioria dos pesquisadores acreditarem que **ZPP** é diferente de **EXP**, provar essa afirmação ainda é um dos problemas mais difíceis em aberto na área de Complexidade Computacional, que necessitaria de técnicas não relativizantes (MURRAY; WILLIAMS, 2014).

5.2 ISOMORFISMO DE GRAFOS

Um *isomorfismo* (WEST, 2000) entre dois grafos G e H é uma bijeção $f : V(G) \to V(H)$ tal que $\overline{uv} \in E(G)$ se e somente se $\overline{f(u)f(v)} \in E(H)$. Dizemos que G é *isomorfo* a H, escrito $G \cong H$, se existe um isomorfismo entre G e H. O problema de Isomorfismo de Grafos pode ser escrito como um problema de decisão da seguinte maneira:

$$\text{GI} = \{\langle G, H\rangle \,;\ \text{os grafos } G \text{ e } H \text{ são isomorfos}\}$$

Como discutido anteriormente, alguns pesquisadores da área de Complexidade Computacional acreditam que a linguagem GI é **NP**-intermediária (SCHÖNING, 1988). O melhor algoritmo até o momento para o problema é de tempo quasi-polinomial (ou então $O(n^{\log n})$) (BABAI, 2015). Além disso, caso o problema de isomorfismo de grafos seja **NP**-completo a hierarquia polinomial colapsa ao segundo nível (BOPPANA et al., 1987).

5.3 RESÍDUO QUADRÁTICO

O problema do resíduo quadrático foi o primeiro para o qual foi mostrado existir um protocolo de conhecimento zero perfeito do tipo "v-bit" (GOLDWASSER et al., 1989). Além disso, o problema encontra aplicação na área de criptografia, através do protocolo Goldwasser-Micali (GOLDWASSER; MICALI, 1982).

A definição do problema é a seguinte: dados como entrada (z, n), onde $z \in \mathbb{Z}_n^*$, responder se z é um resíduo quadrático módulo n, ou seja, se existe $x \in \mathbb{Z}_n^*$ tal que $z = x^2 \pmod{n}$. Sua definição através de um problema de decisão é a seguinte:

$$\text{RQUAD} = \{\langle z, n\rangle \,;\, z \text{ é um resíduo quadrático módulo } n\}$$

Note que o conjunto dos resíduos quadráticos módulo um inteiro n, chamado de RQ_n, é um grupo com multiplicação. Dessa forma, se $x, y \in RQ_n$, então $xy \in RQ_n$. Além

disso, se $y \in RQ_n$ e r é um elemento aleatório de RQ_n, então yr também é um elemento aleatório de RQ_n. Por último, se $y \in RQ_n$ e $x \notin RQ_n$, então $xy \notin RQ_n$.

A função $f(x, n) = x^2 \pmod{n}$ é uma função candidata a ser unidirecional, e por esse motivo há um consenso de que o problema do resíduo quadrático é intratável (ARORA; BARAK, 2009). Ainda assim o problema é redutível ao problema da fatoração, pois conhecer os fatores de n permite resolver o problema em tempo polinomial (GOLDWASSER et al., 1989). Por último, o problema possui algoritmo subexponencial e está contido em $\mathbf{NP} \cap \mathbf{coNP}$, o que torna pouco provável a **NP**-completude desse.

5.4 LOGARITMO DISCRETO

O problema do logaritmo discreto possui diversas similaridades com o problema do resíduo quadrático, pois ambos os problemas são definidos em torno de grupos multiplicativos. O problema também possui um protocolo "v-bit" de conhecimento zero perfeito (MALKA, 2008) e sua dificuldade é a base do protocolo de criptografia Diffie-Hellman (DIFFIE; HELLMAN, 1976).

É mais comum definir o problema do logaritmo discreto como um problema de busca, da seguinte maneira: dados (z, b, n) onde $z, b \in \mathbb{Z}_n$ e n é um número primo, deve-se encontrar x tal que $z = b^x \pmod{n}$. Note que x pode não existir caso b não seja um gerador de $\mathbb{Z}_n$. Porém, quando tratamos da complexidade do problema, é interessante definir uma versão de decisão:

$$\text{LOGD} = \{\langle z, b, n, k\rangle \,; n \text{ é primo e existe } x \leq k \text{ tal que } z = b^x \pmod{n}\}$$

É fácil verificar que resolver a versão de busca do problema resolve a versão de decisão, por outro lado, com um oráculo para a versão de decisão podemos fazer uma busca binária com os valores de k até encontrar o valor x e resolver a versão de busca, mostrando que ambas são equivalentes.

De forma semelhante ao problema do resíduo quadrático, temos as seguintes características: denotando por $H \subseteq \mathbb{Z}_n$ o subgrupo gerado por b, observamos que se $x, y \in H$, então $xy \in H$. Além disso, se $y \in H$ e r é um elemento aleatório de H, então yr também é um elemento aleatório de H. Por último, se $y \in H$ e $x \notin H$, então $xy \notin H$.

A função $f(b, x, n) = b^x \pmod{n}$ também é uma função candidata a ser unidirecional (ARORA; BARAK, 2009). Além disso, o problema também está em $\mathbf{NP} \cap \mathbf{coNP}$ e possui algoritmo subexponencial, o que o torna um candidato a ser **NP**-intermediário.

6 O PODER DO PROBLEMA DE MINIMIZAÇÃO DE CIRCUITOS

Neste capítulo apresentamos o poder computacional do problema de minimização de circuitos, com o objetivo de mostrar como um oráculo para o problema pode ser usado para obter algoritmos aleatorizados polinomiais com oráculo para vários problemas candidatos a **NP**-intermediários.

O problema MCSP possui fortes conexões com um tipo específico de complexidade de Kolmogorov limitada em tempo. Antes de explorar essas conexões, porém, apresentamos a definição formal da medida clássica de complexidade de Kolmogorov em relação à uma MT universal U (LI; VITÁNYI, 1997):

$$\mathrm{K}_U(x) = \{min\,|d| \in \{0,1\}^*; U(d) = x\}$$

De maneira informal, a medida de complexidade de Kolmogorov de uma string x nos diz quanta informação a string x possui. Isso ocorre pois a descrição[1] d usada por U deve ser suficiente para construir a string x do zero. Porém, como a descrição d de uma string x pode representar uma máquina M e uma entrada w tal que $M(w) = x$ seja uma computação ineficiente, faz sentido pensar em versões de K_U em que há um limite de tempo para a máquina M.

Uma maneira de formalizar esse conceito de limite de tempo é a partir da medida de complexidade KT. Na definição a seguir, a notação U^d indica que a máquina U possui acesso indexado à string d, isso é, U possui uma fita de leitura possuindo d e uma fita de endereçamento. U pode então escrever um endereço i na fita endereçamento e obter em um passo computacional o i-ésimo bit de d. Definimos então KT (ALLENDER et al., 2006):

Definição 6.1. (ALLENDER et al., 2006) (Medida de complexidade KT) Seja U uma

[1] Como U é uma MT universal, a descrição d contém uma MT M e uma entrada w.

MT e $x \in \{0,1\}^*$ uma string, definimos:

$$\mathrm{KT}_U(x) = \{\min|d| + t; \forall b \in \{0,1,*\} \forall i \leq |x|+1; U^d(i,b) \text{ aceita em } t \text{ passos sse } x_i = b\}.$$

Ou seja, a máquina U, com acesso aleatório a descrição d, deve reconhecer corretamente os símbolos de x dado seu índice. Como esse mecanismo não determina o tamanho de x, assumimos que para a posição $i = |x| + 1$ o símbolo correto é $*$. Para simplificar, podemos fixar uma Máquina de Turing Universal U qualquer e definir $\mathrm{KT}(x) = \mathrm{KT}_U(x)$, pois a medida KT pode sofrer um *overhead* no máximo logarítmico caso a máquina U seja substituída por alguma outra (ALLENDER et al., 2006).

KT é definido dessa maneira para permitir tempos de execução sublineares para U, o que é de especial interesse quando exploramos sua conexão com o problema MCSP. Para visualizar esse fato, observamos a complexidade da string $x = 0001^{10000}$ (a string constituída por três zeros seguidos por 10000 uns). Certamente uma descrição como a seguinte é suficiente para reconhecer os bits de x:

Na entrada (i, b):
Se $i \leq 3$, então aceite se $b = 0$.
Se $3 \leq i \leq 10003$, então aceite se $b = 1$.
se $i = 10004$, então aceite se $b = *$.

Percebe-se que para executar essa descrição não são necessários tantos passos quanto o tamanho de x, o que seria necessário caso devêssemos produzir a string x em sua totalidade, como ocorre com a medida $\mathrm{K}_U(x)$.

Outra observação importante em relação à medida de complexidade KT é a de que as strings x obtidas através de geradores pseudoaleatórios possuem $\mathrm{KT}(x)$ pequeno (ALLENDER et al., 2006). Isso ocorre pois se a string x foi gerada por um gerador pseudoaleatório G com semente k, podemos descrever x através de G e k, o que impõe um limite superior forte para o valor de $\mathrm{KT}(x)$.

Em relação à conexão com MCSP, considerando que a string x representa a tabela verdade de uma função booleana f_x, temos que o tamanho do menor circuito que computa f_x é aproximadamente o valor de $\mathrm{KT}(x)$. Sendo s o tamanho desse circuito e $|x| = m$, temos (ALLENDER; DAS, 2014):

$$\left(\frac{s}{\log m}\right)^{\frac{1}{4}} \leq \mathrm{KT}(x) \leq \mathcal{O}(s^2(\log s + \log\log m)).$$

Isso significa que uma máquina com oráculo para MCSP pode tomar como entrada a string x e aceitar se e somente se f_x possui circuitos maiores que, por exemplo, $\sqrt{|x|}$. Isso é interessante pois garantimos que essa máquina aceita a maior parte das strings (pois quase todas as funções booleanas precisam de circuitos grandes para serem computadas (SHANNON, 1949b)), porém não aceita nenhuma string x com $\mathrm{KT}(x)$ pequeno, o que torna essa máquina um ótimo teste para distinguir dentre a distribuição uniforme e uma distribuição gerada por um gerador pseudoaleatório (ALLENDER; DAS, 2014).

Esse poder de distinguir distribuições geradas por geradores pseudoaleatórios da distribuição uniforme, devido à forte e clássica conexão entre geradores pseudoaleatórios e funções irreversíveis (HÄSTAD et al., 1999), permite à uma máquina com oráculo para MCSP inverter funções unidirecionais em uma fração polinomial de suas entradas, como indica o teorema a seguir:

Teorema 6.1. *(ALLENDER et al., 2006) Seja L uma linguagem de densidade polinomial*[2] *tal que para algum $\epsilon > 0$ e para todo $x \in L$, $KT(x) \geq |x|^{\epsilon}$. Seja $f(y, x)$ uma função computável em tempo polinomial no tamanho de x. Então existe uma MTP de tempo polinomial com oráculo N^L e um polinômio q tal que:*

$$\Pr_{x \in \{0,1\}^n, s}[f(y, N^L(y, f(y, x), s)) = f(y, x)] \geq \frac{1}{q(n)}$$

A string s acima representa as escolhas aleatórias de N^L. Pelas características já apontadas de MCSP, podemos utilizar um oráculo para esse problema no lugar da linguagem L do Teorema 6.1. A maior parte dos algoritmos que utilizam oráculo para MCSP dependem da ocorrência de pelo menos um sucesso da execução da máquina N^{MCSP}, portanto apresentamos o seguinte lema que irá facilitar as provas de complexidade desses algoritmos:

Lema 6.1. *Se a probabilidade de sucesso de uma execução da MTP M é de $1/x$, então repetir a execução de M $2x$ vezes independentes garante que a probabilidade de que haja pelo menos um sucesso seja maior que $2/3$.*

Demonstração. Definimos a variável aleatória $X_i \in \{0, 1\}$ que representa se foi obtido um sucesso na iésima execução de M. Em especial, $E[X_i] = 1/x$, $X = \sum_{i=1}^{2x} X_i$ e $\mu = \sum_{i=1}^{2x} E[X_i]$. Dessa forma, através do limitante de Chernoff temos que a probabilidade de

[2] L possui pelo menos $2^n/n^k$ strings para cada tamanho n e para algum k.

que nenhuma das $2x$ tentativas obtenha sucesso é de:

$$\Pr[X \leq 0] = \Pr[X \leq \mu - 1\mu] \leq e^{-1\mu} = e^{-2} < \frac{1}{3}$$

Dessa forma a probabilidade de que haja pelo menos um sucesso é maior que $^2/_3$.

□

Esse poder de inverter funções polinomiais em uma fração polinomial das entradas permite à uma máquina com oráculo para MCSP ser utilizada para obter algoritmos aleatorizados para vários problemas candidatos à **NP**-intermediários, como isomorfismo de grafos e até mesmo qualquer problema da classe **SZK**. (ALLENDER; DAS, 2014). Apresentamos as demonstrações dessas proposições a seguir:

Teorema 6.2. *(ALLENDER; DAS, 2014)* $GI \in \boldsymbol{RP}^{MCSP}$.

Demonstração. Dados como entrada G_0 e G_1, devemos responder se $G_0 \cong G_1$. Primeiramente definimos a função computável em tempo polinomial $f(G, \pi) = f_G(\pi)$, que recebe como entrada um grafo G de n vértices e uma permutação $\pi \in S_n$ e tem como saída $\pi(G)$, ou então o resultado da aplicação da permutação π ao grafo G.

Pelo Teorema 6.1, existe uma MTP de tempo polinomial com oráculo N^{MCSP} e um polinômio q tal que para qualquer n e G:

$$\Pr_{\pi \in S_n, s}[f_G(N^{\text{MCSP}}(G, f_G(\pi), s)) = f_G(\pi)] \geq \frac{1}{q(n)}$$

onde π é escolhido de maneira aleatória uniforme e s denota a sequência de escolhas aleatórias de N.

O algoritmo então, tendo como entrada (G_0, G_1), executa os seguintes passos $2q(n)$ vezes independentes:

(1) Sorteie aleatoriamente uma permutação π e uma string s.
(2) Compute $H = f_{G_0}(\pi) = \pi(G_0)$.
(3) Execute $N^{\text{MCSP}}(G_1, H, s)$ e obtenha a saída π'.
(4) Reporte sucesso se $\pi'(G_1) = \pi(G_0)$.
O algoritmo então aceita se pelo menos uma das $2q(n)$ tentativas independentes obtiver sucesso.

Observe que se $G_0 \ncong G_1$, então a probabilidade de sucesso de cada execução é zero, pois não existe π' tal que $\pi'(G_1) = \pi(G_0)$. Por outro lado, se $G_0 \cong G_1$, então o grafo

$H = \pi(G_0)$ aparece na imagem de f_{G_1}, e a probabilidade de sucesso de cada execução é de $1/q(n)$. Pelo Lema 6.1 temos que a probabilidade de que o algoritmo responda corretamente nesse caso é maior que $2/3$. □

Teorema 6.3. *(ALLENDER; DAS, 2014)* $\boldsymbol{SZK \subseteq BPP^{MCSP}}$

Demonstração. Como visto na seção 4.4, basta mostrar que PIID $\in \mathbf{BPP}^{\mathrm{MCSP}}$. Definimos a função $f(C, x) = f_C(x)$, que recebe um circuito Booleano C com entrada de tamanho m e uma string $x \in \{0,1\}^m$, e tem como saída $C(x)$, ou então a resultado do circuito C com a entrada x.

Pelo Teorema 6.1, existe uma MTP de tempo polinomial com oráculo N^{MCSP} e um polinômio q tal que para qualquer m e C:

$$\Pr_{x\in\{0,1\}^m, s}[f_C(N^{\mathrm{MCSP}}(C, f_C(x), s)) = f_C(x)] \geq \frac{1}{q(m)}$$

onde x é escolhido de maneira aleatória uniforme e s denota a sequência de escolhas aleatórias de N.

O algoritmo então, com a entrada (n, D_0, D_1), executa os seguintes passos n^ℓ vezes independentes (para um ℓ determinado posteriormente):

(1) Sorteie aleatoriamente uma string $r \in_R \{0,1\}^m$ e uma string aleatória s.
(2) Compute $y = f_{D_0}(r) = D_0(r)$.
(3) Execute $N^{\mathrm{MCSP}}(D_1, y, s)$ e obtenha a string $x \in \{0,1\}^m$.
(4) Reporte sucesso se $D_0(r) = D_1(x)$.
O algoritmo então aceita se pelo menos $\log(n)$ das n^ℓ tentativas independentes obtiverem sucesso.

Se (n, D_0, D_1) é uma instância "Não" de PIID, então a probabilidade de que uma execução obtenha sucesso é de no máximo $1/2^n$. Portanto, para n suficientemente grande, a quantidade esperada de sucessos das n^ℓ execuções é de no máximo $n^\ell/2^n < 1$. Definimos a variável aleatória $X_i \in \{0,1\}$ que representa se foi obtido um sucesso na iésima execução do algoritmo. Em especial, $X = \sum_{i=1}^{n^\ell} X_i$ e fazendo $\mu = 1$ temos pelo limitante de Chernoff:

$$\Pr[X \geq \log(n)] < \Pr[X \geq (1 + (\log(n) - 1))\mu] \leq \frac{e^{\log n - 1}}{\log n^{\log n}}$$

para n suficientemente grande, essa probabilidade é menor que $1/3$.

Já se (n, D_0, D_1) é uma instância "Sim" de PIID, então D_0 e D_1 possuem distância

estatística no máximo $1/2^n$. Queremos então encontrar a probabilidade p de sucesso de cada execução do algoritmo, note que:

$$p = \Pr_{x\in\{0,1\}^m,s}[f_{D_1}(N^{\text{MCSP}}(D_1, f_{D_0}(x), s)) = f_{D_0}(x)]$$

$$= \sum_z \Pr_{x\in\{0,1\}^m,s}[f_{D_1}(N^{\text{MCSP}}(D_1, z, s) = z | z = f_{D_0}(x)]\Pr[z = f_{D_0}(x)]$$

$$= \sum_z \Pr_{x\in\{0,1\}^m,s}[f_{D_1}(N^{\text{MCSP}}(D_1, z, s) = z | z = f_{D_1}(x)]\Pr[z = f_{D_0}(x)]$$

Além disso:

$$p' = \Pr_{x\in\{0,1\}^m,s}[f_{D_1}(N^{\text{MCSP}}(D_1, f_{D_1}(x), s)) = f_{D_1}(x)]$$

$$= \sum_z \Pr_{x\in\{0,1\}^m,s}[f_{D_1}(N^{\text{MCSP}}(D_1, z, s) = z | z = f_{D_1}(x)]\Pr[z = f_{D_1}(x)]$$

Fazendo $p' - p$ temos:

$$\sum_z \Pr_{x\in\{0,1\}^m,s}[f_{D_1}(N^{\text{MCSP}}(D_1, z, s) = z | z = f_{D_1}(x)] \cdot (\Pr[z = f_{D_1}(x)] - \Pr[z = f_{D_0}(x)])$$

$$\leq \sum_z 1 \cdot (\Pr[z = f_{D_1}(x)] - \Pr[z = f_{D_0}(x)])$$

$$\leq \frac{1}{2^n}$$

Como $p' > 1/q(m) > 1/q(n^k)$, temos que $p \geq 1/q(n^k) - 1/2^n$. A quantidade esperada de sucessos é então de no mínimo $n^\ell(1/q(n^k) - 1/2^n)$. Escolhendo ℓ suficientemente maior que $q(n^k)$ garantimos que essa quantidade esperada seja pelo menos n. Definimos a variável aleatória $X_i \in \{0, 1\}$ que representa se foi obtido um sucesso na iésima execução do algoritmo. Em especial, $X = \sum_{i=1}^{n^\ell} X_i$ e fazendo $\mu = n$ temos pelo limitante de Chernoff:

$$\Pr[X \leq \log n] \leq \Pr\left[X \leq \frac{n}{2}\right] \leq \frac{e^{-\mu/2}}{1/\sqrt{2}} = \frac{\sqrt{2}}{e^{n/2}}$$

Para n suficientemente grande essa probabilidade fica abaixo de $1/3$. Portanto, a probabilidade de que pelo menos $\log n$ das n^ℓ tentativas obtenham sucesso é maior que $2/3$. □

Podemos utilizar ainda o mesmo algoritmo do Teorema 6.3 para o problema $\text{SD}^{0,1}$, e nesse caso o algoritmo pode errar sua resposta somente no caso em que a instância é uma instância "Sim" do problema. Como $\text{SD}^{0,1}$ é completo para os problemas da classe **PZK** com protocolos "v-bit", essa classe está contida em $\mathbf{RP}^{\text{MCSP}}$ (ALLENDER; DAS, 2014).

7 ALGORITMOS ALEATORIZADOS POLINOMIAIS COM ORÁCULO PARA MCSP

Neste capítulo apresentamos a principal contribuição deste trabalho, algoritmos aleatorizados polinomiais para os problemas do resíduo quadrático (RQUAD) e do logaritmo discreto (LOGD). Combinando o Teorema 6.3 com a informação de que ambos os problemas possuem protocolos de conhecimento zero do tipo "v-bit", temos que os dois problemas estão em $\mathbf{RP}^{\mathrm{MCSP}}$. Além disso, para o problema do RQUAD, como esse é redutível para o problema da fatoração, que está em $\mathbf{ZPP}^{\mathrm{MCSP}}$ (ALLENDER et al., 2006), temos que RQUAD $\in \mathbf{ZPP}^{\mathrm{MCSP}}$.

Ainda assim, como esses resultados são consequências de reduções, não fica necessariamente claro o papel que um oráculo para MCSP tem na resolução desses dois problemas. Os algoritmos apresentados Neste capítulo são bastante diretos e simples, deixando claro esse papel.

7.1 ALGORITMO PARA O PROBLEMA DO RESÍDUO QUADRÁTICO

Para o problema do resíduo quadrático definimos a função computável em tempo polinomial $f : \mathbb{Z}_n^* \times \mathbb{Z}_n^* \to \mathbb{Z}_n^*$, que em conjunto com a máquina N^{MCSP} nos permitirá encontrar um certificado para a instância de RQUAD. Definimos $f(u, v) = f_u(v) = uv^2 \pmod n$.

Pelo Teorema 6.1, sabemos que existe uma MTP polinomial com oráculo N^{MCSP} e um polinômio q tal que:

$$\Pr_{v \in \mathbb{Z}_n^*, s}[f_u(N^{\mathrm{MCSP}}(u, f_u(v), s)) = f_u(v)] \geq \frac{1}{q(|n|)}$$

Isso é, dado um resultado $f_u(v) = y$, a máquina N^{MCSP} consegue encontrar um elemento $t \in \mathbb{Z}_n^*$ tal que $f_u(v) = f_u(t)$ com probabilidade igual ou maior que $1/q(|n|)$, dada em relação às escolhas aleatórias de N^{MCSP} (a string s) e de v.

Dada a entrada (z, n), a ideia principal do algoritmo é computar $y = f_z(r)$ para um valor r escolhido de maneira aleatória uniforme em $\mathbb{Z}_n^*$, e então executar a máquina N^{MCSP} com a entrada (r^2, y, s). Dessa forma, com probabilidade $1/q(|n|)$ encontramos um elemento x tal que $y = r^2x^2 \pmod n$, porém como $y = zr^2 \pmod n$, temos $z = x^2 \pmod n$, o que nos permite verificar que a instância é verdadeira.

Algoritmo para RQUAD: Dados como entrada (z, n) tal que $z \in \mathbb{Z}_n^*$, o algoritmo proposto executa independentemente os seguintes passos $2q(|n|)$ vezes:

(1) Sorteie aleatoriamente um número $r \in \mathbb{Z}_n^*$ e uma string aleatória s.
(2) Compute $y = f_z(r) = zr^2 \pmod n$.
(3) Execute $N^{\text{MCSP}}(r^2, y, s)$ e obtenha o número $x \in \mathbb{Z}_n^*$.
(4) Reporte sucesso se $z = x^2 \pmod n$.
O algoritmo aceita se pelo menos uma das tentativas obtiver sucesso.

Observe que se $z \notin RQ_n$, então não haverá nenhum sucesso, pois y não aparece na imagem de f_{r^2} já que é o produto de um resíduo e um não resíduo. Por outro lado, se $z \in RQ_n$ temos probabilidade $1/q(|n|)$ por execução de encontrar x tal que $z = x^2 \pmod n$. Pelo Lema 6.1 temos que a probabilidade de que o algoritmo responda corretamente nesse caso é maior que $2/3$, portanto esse é da classe $\mathbf{RP}^{\text{MCSP}}$.

7.2 ALGORITMO PARA O PROBLEMA DO LOGARITMO DISCRETO

O algoritmo apresentado para o problema do logaritmo discreto é semelhante ao apresentado para o problema do resíduo quadrático. Isso se deve ao fato de que ambos os problemas possuem estrutura semelhante, pois são definidos sobre grupos finitos e além disso envolvem a inversão de uma função candidata a unidirecional.

Para o problema do logaritmo discreto, definimos a função computável em tempo polinomial $f : \mathbb{Z}_n \times \mathbb{Z}_n \to \mathbb{Z}_n$ como $f(u, v) = f_u(v) = ub^v \pmod n$. Sabemos pelo Teorema 6.1 que existe uma MTP polinomial com oráculo N^{MCSP} e um polinômio q tal que:

$$\Pr_{v \in \mathbb{Z}_n, s}[f_u(N^{\text{MCSP}}(u, f_u(v), s)) = f_u(v)] \geq \frac{1}{q(|v|)}$$

Ou seja, dado um resultado $f_u(v) = y$, a máquina N^{MCSP} consegue encontrar um elemento $t \in \mathbb{Z}_n$ tal que $f_u(v) = f_u(t)$ com probabilidade igual ou maior que $1/q(|n|)$, dada em relação às escolhas aleatórias de N^{MCSP} (a string s) e de v.

A ideia do algoritmo é semelhante, utiliza-se o poder de N^{MCSP} para inverter a

função f em uma fração polinomial das entrada, trocando o parâmetro fixo de f, de forma a encontrar x tal que $b^x = z \pmod{n}$.

Algoritmo para LOGD: Dados (z, b, n) tal que $z, b \in \mathbb{Z}_n$, primeiro testamos se n é primo em tempo polinomial (AGRAWAL et al., 2004), e depois executamos independentemente os seguintes passos $2q(|n|)$ vezes:

(1) Sorteie aleatoriamente um número $r \in \mathbb{Z}_n$ e uma string aleatória s.
(2) Compute $y = f_z(r) = zb^r \pmod{n}$.
(3) Execute $N^{\text{MCSP}}(b^r, y, s)$ e obtenha o número $x \in \mathbb{Z}_n$.
(4) Reporte sucesso se $z = b^x \pmod{n}$.
O algoritmo então retorna x se pelo menos uma das tentativas obtém sucesso.

Caso z não pertença ao subgrupo $H \subseteq \mathbb{Z}_n$ gerado por b, então não haverá nenhum sucesso, pois y não aparece na imagem de f_{b^r}. Por outro lado, se $z \in H$ temos probabilidade $1/q(|n|)$ por execução de encontrar x tal que $z = b^x \pmod{n}$. Novamente, pelo Lema 6.1, temos que esse algoritmo é da classe $\mathbf{RP}^{\text{MCSP}}$.

7.3 CONCLUSÃO E TRABALHOS FUTUROS

O problema de minimização de circuitos, apesar de provavelmente não ser **NP**-completo, aparenta ser bastante complexo. Um oráculo para o problema é capaz de resolver diversos problemas candidatos a **NP**-intermediários em tempo polinomial com alta probabilidade. Percebemos, porém, algumas peculiaridades no desenvolvimento do trabalho.

Os problemas para os quais foi possível mostrar algoritmos aleatorizados polinomiais com oráculo para MCSP parecem compartilhar algumas propriedades. Notavelmente, problemas como isomorfismo de grafos, resíduo quadrático e logaritmo discreto compartilham o fato de serem definidos sobre estruturas de grupos. Os algoritmos apresentados para esses problemas também apresentam uma estrutura bastante semelhante.

Outra característica dos problemas que parece importante para aplicar as técnicas utilizadas é a possibilidade de construir uma instância única deterministicamente através de um certificado (para a instância construída). No caso do resíduo quadrático, por exemplo, basta escolher um elemento aleatório em $\mathbb{Z}_n^*$ e elevá-lo ao quadrado módulo n para obter uma instância aleatória do problema.

No entanto, existem problemas que também são candidatos a **NP**-intermediários

porém aparentam não possuir estrutura de grupos nem a possibilidade de gerar instâncias aleatórias deterministicamente através de um certificado. Um exemplo é o problema do conjunto dominante mínimo em torneios (DOMT) (MEGIDDO; VISHKIN, 1988). Apesar de o problema possuir algoritmo quasi-polinomial ($O(n^{\log n})$), não aparenta possuir as características citadas.

Levando em consideração o poder de um oráculo para MCSP, nos perguntamos se não há como utilizar esse oráculo para obter algoritmos aleatorizados polinomiais para esses outros problemas. Em especial, nos perguntamos se DOMT $\in \mathbf{RP}^{\mathrm{MCSP}}$ ou então pelo menos DOMT $\in \mathbf{BPP}^{\mathrm{MCSP}}$.

Além disso, nos perguntamos se é possível obter reduções mais diretas para MCSP dos problemas apresentados. Por exemplo, imaginamos que os três problemas estudados possam estar contidos em $\mathbf{P}^{\mathrm{MCSP}}$, porém as técnicas atuais aparentam não permitir realizar esse tipo de redução para o problema.

REFERÊNCIAS

ADLEMAN, L. Two theorems on random polynomial time. In: **Proceedings of the 19th Annual Symposium on Foundations of Computer Science**. Washington, DC, USA: IEEE Computer Society, 1978. (SFCS '78), p. 75–83.

AGRAWAL, M.; KAYAL, N.; SAXENA, N. PRIMES is in P. **Ann. Math. (2)**, Princeton University, Mathematics Department, Princeton, NJ; Mathematical Sciences Publishers (MSP), Berkeley, CA, v. 160, n. 2, p. 781–793, 2004. ISSN 0003-486X; 1939-8980/e.

ALLENDER, E. et al. Power from random strings. **SIAM Journal on Computing**, v. 35, n. 6, p. 1467–1493, 2006.

ALLENDER, E.; DAS, B. Zero knowledge and circuit minimization. **Electronic Colloquium on Computational Complexity (ECCC)**, v. 21, p. 68, 2014.

ARORA, S.; BARAK, B. **Computational Complexity: A Modern Approach**. 1st. ed. New York, NY, USA: Cambridge University Press, 2009. ISBN 0521424267, 9780521424264.

BABAI, L. Graph isomorphism in quasipolynomial time. **CoRR**, abs/1512.03547, 2015.

BAKER, T.; GILL, J.; SOLOVAY, R. Relativizations of the $\mathcal{P} =? \mathcal{NP}$ question. **SIAM J. Comput.**, v. 4, p. 431–442, 1975.

BEN-OR; GUTFREUND. Trading help for interaction in statistical zero-knowledge proofs. **Journal of Cryptology**, v. 16, n. 2, p. 95–116, 2003. ISSN 1432-1378.

BOPPANA, R. B.; HASTAD, J.; ZACHOS, S. Does co-np have short interactive proofs? **Inf. Process. Lett.**, Elsevier North-Holland, Inc., Amsterdam, The Netherlands, The Netherlands, v. 25, n. 2, p. 127–132, maio 1987. ISSN 0020-0190.

COBHAM, A. The intrinsic computational difficulty of functions. In: **Procedings of the 1964 Congress for Logic, Methodology and Philosophy of Science**. [S.l.]: Elsevier/North-Holland, 1964. p. 24–30.

COOK, S. A. The complexity of theorem-proving procedures. In: **Proceedings of the Third Annual ACM Symposium on Theory of Computing**. New York, NY, USA: ACM, 1971. (STOC '71), p. 151–158.

COOK, S. A. A hierarchy for nondeterministic time complexity. **J. Comput. Syst. Sci.**, Academic Press, Inc., Orlando, FL, USA, v. 7, n. 4, p. 343–353, ago. 1973. ISSN 0022-0000.

DIFFIE, W.; HELLMAN, M. New directions in cryptography. **IEEE Transactions on Information Theory**, v. 22, n. 6, p. 644–654, Nov 1976. ISSN 0018-9448.

GILL, J. T. Computational complexity of probabilistic turing machines. In: **Proceedings of the Sixth Annual ACM Symposium on Theory of Computing**. New York, NY, USA: ACM, 1974. (STOC '74), p. 91–95.

GOLDREICH, O.; MICALI, S.; WIGDERSON, A. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. **J. ACM**, ACM, New York, NY, USA, v. 38, n. 3, p. 690–728, jul. 1991. ISSN 0004-5411.

GOLDWASSER, S.; MICALI, S. Probabilistic encryption & how to play mental poker keeping secret all partial information. In: **Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing**. New York, NY, USA: ACM, 1982. (STOC '82), p. 365–377. ISBN 0-89791-070-2.

GOLDWASSER, S.; MICALI, S.; RACKOFF, C. The knowledge complexity of interactive proof systems. **SIAM J. Comput.**, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, v. 18, n. 1, p. 186–208, fev. 1989. ISSN 0097-5397.

HARTMANIS, J.; STEARNS, R. E. On the computational complexity of algorithms. **Transactions of the American Mathematical Society**, v. 117, p. 285–306, 1966.

HÄSTAD, J. et al. A pseudorandom generator from any one-way function. **SIAM Journal on Computing**, v. 28, n. 4, p. 1364–1396, 1999.

HENNIE, F. C.; STEARNS, R. E. Two-tape simulation of multitape turing machines. **J. ACM**, ACM, New York, NY, USA, v. 13, n. 4, p. 533–546, out. 1966. ISSN 0004-5411.

HIRAHARA, S.; WATANABE, O. Limits of minimum circuit size problem as oracle. **Electronic Colloquium on Computational Complexity (ECCC)**, v. 22, p. 198, 2015.

HOPCROFT, J. E.; MOTWANI, R.; ULLMAN, J. D. **Introduction to automata theory, languages, and computation - (2. ed.)**. [S.l.]: Addison-Wesley-Longman, 2001. (Addison-Wesley series in computer science). ISBN 978-0-201-44124-6.

KARP, R. Reducibility among combinatorial problems. In: MILLER, R.; THATCHER, J. (Ed.). **Complexity of Computer Computations**. New York: Plenum Press, 1972.

KARP, R.; LIPTON, R. Turing machines that take advice. **L' Ensignement Mathématique**, v. 28, p. 191—-210, 1982.

LADNER, R. E. On the structure of polynomial time reducibility. **J. ACM**, ACM, New York, NY, USA, v. 22, n. 1, p. 155–171, jan. 1975. ISSN 0004-5411.

LAUTEMANN, C. {BPP} and the polynomial hierarchy. **Information Processing Letters**, v. 17, n. 4, p. 215 – 217, 1983. ISSN 0020-0190.

LEVIN, L. A. Universal sequential search problems. **Problems of Information Transmission**, v. 9, n. 3, p. 265–266, 1973.

LI, M.; VITÁNYI, P. **An introduction to Kolmogorov complexity and its applications**. 2. ed. [S.l.]: Springer-Verlag, 1997.

LUND, C. et al. Algebraic methods for interactive proof systems. **J. ACM**, ACM, New York, NY, USA, v. 39, n. 4, p. 859–868, out. 1992. ISSN 0004-5411.

MALKA, L. **A Study of Perfect Zero-knowledge Proofs**. Tese (Doutorado) — University of Victoria, Victoria, B.C., Canada, 2008. AAINR47335.

MEGIDDO, N.; VISHKIN, U. On finding a minimum dominating set in a tournament. **Theoretical Computer Science**, v. 61, n. 2, p. 307 – 316, 1988. ISSN 0304-3975.

MEYER, A. R.; STOCKMEYER, L. J. The equivalence problem for regular expressions with squaring requires exponential space. In: **Proceedings of the 13th Annual Symposium on Switching and Automata Theory (Swat 1972)**. Washington, DC, USA: IEEE Computer Society, 1972. (SWAT '72), p. 125–129.

MURRAY, C.; WILLIAMS, R. On the (non) np-hardness of computing circuit complexity. **Electronic Colloquium on Computational Complexity (ECCC)**, v. 21, p. 164, 2014.

PAPADIMITRIOU, C. M. **Computational complexity**. Reading, Massachusetts: Addison-Wesley, 1994. ISBN 0201530821.

SCHÖNING, U. Graph isomorphism is in the low hierarchy. **J. Comput. Syst. Sci.**, Academic Press, Inc., Orlando, FL, USA, v. 37, n. 3, p. 312–323, dez. 1988.

SHAMIR, A. IP = PSPACE. **J. ACM**, ACM, New York, NY, USA, v. 39, n. 4, p. 869–877, out. 1992. ISSN 0004-5411.

SHANNON, C. E. Communication theory of secrecy systems. **The Bell System Technical Journal**, v. 28, n. 4, p. 656–715, Oct 1949. ISSN 0005-8580.

SHANNON, C. E. The synthesis of two terminal switching circuits. **j-BELL-SYST-TECH-J**, v. 28, n. 1, p. 59–98, jan. 1949. ISSN 0005-8580.

SIPSER, M. A complexity theoretic approach to randomness. In: **Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing**. New York, NY, USA: ACM, 1983. (STOC '83), p. 330–335. ISBN 0-89791-099-0.

STEARNS, R. E.; HARTMANIS, J.; II, P. M. L. Hierarchies of memory limited computations. In: **SWCT (FOCS)**. Michigan, USA: IEEE Computer Society, 1965. p. 179–190.

STOCKMEYER, L. J.; MEYER, A. R. Word problems requiring exponential time(preliminary report). In: **Proceedings of the Fifth Annual ACM Symposium on Theory of Computing**. New York, NY, USA: ACM, 1973. (STOC '73), p. 1–9.

TURING, A. M. On computable numbers, with an application to the Entscheidungsproblem. **Proceedings of the London Mathematical Society**, v. 2, n. 42, p. 230–265, 1936.

WEST, D. B. **Introduction to Graph Theory**. 2. ed. Delhi, India: Prentice Hall, 2000. ISBN 0130144002.