COMPUTAÇÃO QUÂNTICA E COMPLEXIDADE COMPUTACIONAL

CARLOS HENRIQUE CARDONHA
CRISTINA G. FERNANDES (ORIENTADORA)

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
UNIVERSIDADE DE SÃO PAULO

RESUMO. O estudo de computabilidade e complexidade computacional foi sempre inicialmente motivado e impulsionado por questões matemáticas fundamentais. Nos anos 80, foi introduzido o modelo quântico de computação que, na década passada, se mostrou especialmente adequado para resolver eficientemente problemas matemáticos como o da fatoração de inteiros e o do cálculo do logaritmo discreto de um número. Será apresentado o modelo quântico de computação e algumas relações entre as classes de complexidade clássicas **P**, **NP** e **BPP** e outras com as novas classes de complexidade baseadas no modelo quântico. No final do texto comento sobre o BCC e sua importância na elaboração desse trabalho de iniciação científica.

1. INTRODUÇÃO

Em 1900, em uma palestra marcante no Congresso Internacional de Matemáticos realizado em Paris, Hilbert postulou 23 problemas matemáticos, que tratam de temas diversos em matemática e áreas afins. O décimo problema na lista de Hilbert (*determination of the solvability of a diophantine equation*) pergunta se é possível determinar se uma equação diofantina arbitrária tem ou não solução por meio de um "processo finito":

> *Given a diophantine equation with any number of unknown quantities and with rational numerical coefficients: to devise a process according to which it can be determined by a finite number of operations whether the equation is solvable in rational integers.*

Esse problema pode ser postulado em uma linguagem mais atual como o seguinte: existe um algoritmo que, dada uma equação diofantina, determina se esta tem ou não solução?

Note que a questão postulada por Hilbert precede de décadas a invenção de computadores. Foi apenas nos anos 30 que tais questões foram formuladas

Financiado parcialmente pela FAPESP 03/13236-0.

e tratadas dentro do que ficou depois conhecido como *teoria da computabilidade*. Esta é a parte da teoria da computação especializada em lidar com esse tipo de questão.

Foi nos anos 30, após um trabalho de Gödel em lógica, que a idéia de algoritmo começou a ser formalizada. Gödel [Göd31] introduziu o conceito de *função primitiva recursiva* como uma formalização dessa idéia. Church [Chu33, Chu36] introduziu o λ-*cálculo* e Kleene [Kle36] definiu o conceito de *funções recursivas parciais* e mostrou a equivalência entre esse e o λ-cálculo. Turing [Tur36, Tur37] por sua vez propôs a sua formalização da idéia de algoritmo: as chamadas *máquinas de Turing*. Nesses trabalhos, Turing mostrou também a equivalência do conceito de máquinas de Turing e de funções recursivas parciais de Church. Vale mencionar que o conceito de máquinas de Turing foi independentemente proposto por Post [Pos36], um professor de colegial de Nova Iorque. Cada uma dessas propostas diferentes do conceito de algoritmo é chamada de *modelo de computação*.

Foi Kleene [Kle52] quem chamou de *tese de Church* a afirmação de que todo modelo de computação *razoável* é equivalente ao da máquina de Turing. A afirmação é propositalmente vaga, pois visa capturar mesmo modelos que ainda venham a ser propostos, e cuja natureza não podemos prever. Por *razoável* entende-se um modelo que seja realista, no sentido de poder (mesmo que de maneira aproximada) ser construído na prática.

A teoria da computabilidade no fundo diferencia os problemas *decidíveis* (para os quais existe um algoritmo) dos *indecidíveis* (para os quais não existe um algoritmo). O surgimento dos computadores nas décadas de 30 e 40 aos poucos evidenciou uma diferença entre os problemas decidíveis: muitos parecem ser bem mais difíceis que outros, no sentido de que se conhece apenas algoritmos extremamente lentos para eles. Com isso, surgiu a necessidade de refinar a teoria de computabilidade para tentar explicar essas diferenças. Foi apenas nos anos 60 que a *teoria de complexidade*, que trata de tais questões, tomou corpo, com a formalização da idéia de "algoritmo eficiente", independentemente introduzida por Cobham [Cob65] e Edmonds [Edm65], e a proposta de reduções eficientes entre problemas [Kar72].

Foi nessa época que surgiram as definições das classes de complexidade **P** e **NP** e do conceito de **NP**-*completude*, que captura de certa maneira a dificuldade de se conseguir algoritmos eficientes para certos problemas. Grosseiramente, um problema em **NP** é dito **NP**-*completo* se qualquer outro problema da classe **NP** pode ser reduzido eficientemente a ele. A mais famosa questão na área de teoria da computação é se **P** é ou não igual a **NP**. Se for mostrado que algum problema **NP**-completo está em **P**, então tal questão é resolvida e fica provado que $\mathbf{P} = \mathbf{NP}$.

Um marco na teoria de complexidade é o *teorema de Cook* [Coo71, Lev73], que prova a existência de problemas **NP**-completos. Cook mostrou que o problema conhecido como SAT, de decidir se uma fórmula booleana em forma normal conjuntiva é ou não satisfatível, é **NP**-completo. Após o teorema de Cook e o trabalho de Karp [Kar72], que mostrou que vários outros problemas

conhecidos de otimização combinatória eram **NP**-completos, essa teoria se desenvolveu amplamente, tendo estabelecido a dificuldade computacional de problemas das mais diversas áreas [GJ79].

Um dos problemas mais famosos cuja complexidade continua em aberto, mesmo após várias décadas de esforço da comunidade no sentido de resolvê-lo, é o problema da *fatoração de inteiros*: dado um inteiro, determinar a sua fatoração em números primos. Recentemente, o seu parente próximo, o problema de decidir se um número inteiro é primo ou não, chamado de *problema da primalidade*, teve sua complexidade totalmente definida, com o algoritmo AKS, de Agrawal, Kayal e Saxena [AKS02a, AKS02b]. Esse algoritmo mostra que o problema da primalidade está na classe **P**, resolvendo com isso uma questão em aberto há anos. Não se sabe até hoje, no entanto, se há um algoritmo eficiente para resolver o problema da fatoração de inteiros!

Na verdade, a dificuldade computacional do problema da fatoração de inteiros tem sido usada de maneira crucial em alguns sistemas criptográficos bem-conhecidos. Se for descoberto um algoritmo eficiente para resolver o problema da fatoração, vários sistemas criptográficos importantes seriam quebrados, incluindo o famoso sistema RSA de chave pública, criado por Rivest, Shamir e Adleman [RSA78].

O assunto de nossa iniciação científica — Computação Quântica — trata de um novo modelo de computação, o modelo quântico, que vem levantando questões intrigantes dentro da teoria de complexidade, e pode ter impactos práticos dramáticos no mínimo na área de criptologia. O modelo quântico de computação não infringe a validade da tese de Church, porém questiona a validade de uma versão mais moderna dessa, a chamada *tese de Church estendida*, que diz que todo modelo de computação razoável pode ser simulado *eficientemente* por uma máquina de Turing.

Pode-se dizer que a teoria de computação quântica iniciou-se nos anos 80, quando Feynman [Fey82] observou que um sistema quântico de partículas, ao contrário de um sistema clássico, parece não poder ser simulado eficientemente em um computador clássico e sugeriu um computador que explorasse efeitos da física quântica para contornar o problema. Desde então, até 1994, a teoria de computação quântica desenvolveu-se discretamente, com várias contribuições de Deutsch [Deu85, Deu89], Bernstein e Vazirani [BV97], entre outros, que colaboraram fundamentalmente para a formalização de um modelo computacional quântico.

Foi apenas em 1994 que a teoria recebeu um forte impulso e uma enorme divulgação. Isso deveu-se principalmente ao algoritmo de Shor [Sho94, Sho97], um algoritmo quântico eficiente para o problema da fatoração de inteiros, considerado o primeiro algoritmo quântico combinando relevância prática e eficiência. O algoritmo de Shor é uma evidência de que o modelo computacional quântico proposto pode superar de fato o modelo clássico, derivado das máquinas de Turing. O resultado de Shor impulsionou tanto a pesquisa prática, objetivando a construção de um computador segundo o modelo quântico, quanto a busca por algoritmos criptográficos alternativos e

algoritmos quânticos eficientes para outros problemas difíceis. Essas e várias outras questões, relacionadas tanto com a viabilidade do modelo quântico quanto com as suas limitações, têm sido objeto de intensa pesquisa científica.

Do ponto de vista prático, busca-se descobrir se é ou não viável construir um computador segundo o modelo quântico que seja capaz de manipular números suficientemente grandes. Tal viabilidade esbarra em uma série de questões técnicas e barreiras físicas e tecnológicas. Já se tem notícia de computadores construídos segundo o modelo quântico, mas todos ainda de pequeno porte. Em 2001, por exemplo, foi construído um computador quântico com 7 *qubits* (o correspondente aos bits dos computadores tradicionais). Nesse computador, foi implementado o algoritmo de Shor que, nele, fatorou o número 15. Uma parte dos cientistas da computação acredita que a construção de computadores quânticos de maior porte será possível, enquanto outra parte não acredita nisso.

Do ponto de vista de teoria de complexidade, busca-se estabelecer a relação entre as classes de complexidade derivadas do modelo quântico e as classes de complexidade tradicionais. Também busca-se, claro, estabelecer a complexidade no modelo quântico de problemas bem-conhecidos, ou seja, busca-se por algoritmos quânticos eficientes para outros problemas relevantes.

Nesse trabalho, apresentamos os modelos mais tradicionais de computação e o modelo quântico, definimos as principais classes de complexidade clássicas e quânticas, e a apresentamos alguns resultados de complexidade envolvendo tais classes. A parte do nosso trabalho de iniciação científica submetida por Marcel K. De Carli Silva mostra o resultado algorítmico mais relevante na área de computação quântica: o algoritmo de Shor.

Esperamos que este trabalho dê uma visão do que é esta área nova e intrigante, e das suas potencialidades e dificuldades. O tema é multidisciplinar, no sentido de que depende de uma série de conceitos da mecânica quântica, e empresta a notação usada nessa área, o que dificulta um pouco a apresentação dos conceitos para pessoas de outras áreas, como computação e matemática. Um texto mais completo que estamos preparando dentro dessa iniciação científica, com esses resultados e outros, pode ser encontrado no endereço `http://www.ime.usp.br/~magal/quantum/`.

2. Máquinas de Turing

Agora mudamos de assunto, nos voltando à teoria de complexidade computacional. Os resultados dessa área são usualmente apresentados por meio do mais tradicional modelo de computação — a máquina de Turing (MT), que nada mais é que um computador bastante rudimentar com memória infinita.

Vamos apresentar três variantes desse modelo: a máquina de Turing determinística, a não-determinística e a probabilística. Depois disso, apresentamos a segunda formalização do modelo quântico de computação, a

máquina de Turing quântica, fazendo um paralelo com o modelo clássico de computação. Como observamos na introdução, essa segunda formalização é equivalente à primeira. A demonstração desse fato não é trivial e não será mostrada nesse texto.

2.1. **Máquina de Turing determinística.** Uma máquina de Turing determinística (MTD) é composta por uma central de controle, uma cabeça de leitura e uma fita dividida em células. Essa fita tem um final à esquerda e contém infinitas células à direita.

Cada célula da fita armazena um símbolo pertencente a um conjunto finito Σ. O conjunto Σ é chamado de *alfabeto* da máquina e contém, entre outros, dois símbolos especiais: o símbolo $\sqcup$, chamado de *branco*, e o símbolo $\triangleright$, que fica armazenado o tempo todo na célula mais à esquerda da fita.

A cabeça de leitura da máquina é um apontador móvel para uma determinada célula da fita. O conteúdo dessa célula pode ser lido e alterado pela máquina.

A cada instante, a central de controle da máquina está em um dos estados de um conjunto finito Q de estados. No instante inicial, a máquina começa em um estado particular de Q, chamado de estado *inicial* e denotado por s. Existe ainda um conjunto especial de estados $H \subseteq Q$, chamados de estados *finais*.

Uma MTD funciona da seguinte maneira. Ela recebe como entrada uma cadeia de caracteres em $\big(\Sigma \setminus \{\sqcup, \triangleright\}\big)^*$. Inicialmente a cabeça de leitura aponta para a célula mais à esquerda da fita, e a partir da célula seguinte está armazenada a entrada da máquina, seguida por brancos. A máquina efetua uma seqüência de passos até terminar a execução. Se q é o estado corrente da máquina e q está em H, então ela termina a execução. Do contrário, ela realiza três ações, determinadas pelo par (q, σ), onde σ é o símbolo de Σ contido na célula que está sob a cabeça de leitura.

A primeira ação consiste na alteração do estado da máquina de q para um estado q' em Q. A segunda ação é a escrita de um símbolo σ' na célula que está sob a cabeça de leitura. A terceira ação consiste no deslocamento da cabeça de leitura, que pode se mover uma posição para a esquerda, uma posição para a direita ou pode continuar apontando para a mesma célula.

A cabeça de leitura da fita sempre desloca-se para a direita quando atinge a célula mais à esquerda e nunca altera o conteúdo dessa posição da fita. Por conveniência, assume-se que a máquina não pode escrever o símbolo $\triangleright$ em qualquer posição da fita que não seja a célula mais à esquerda.

A cadeia de caracteres escrita na fita quando a máquina termina a execução, ignorando-se o símbolo $\triangleright$ e os brancos à direita, é a saída da máquina para a entrada em questão.

Formalmente, define-se uma máquina de Turing determinística como uma quíntupla $M := (Q, \Sigma, \delta, s, H)$, onde Q é o conjunto de estados, Σ é o alfabeto de símbolos, δ é uma função de $(Q \setminus H) \times \Sigma$ em $Q \times \Sigma \times \{\leftarrow, \downarrow, \rightarrow\}$, $s \in Q$ é o

estado inicial e $H \subseteq Q$ é o conjunto de estados finais. O conjunto $\{\leftarrow, \downarrow, \rightarrow\}$ descreve os possíveis movimentos da cabeça de leitura da máquina.

A função δ, chamada usualmente de *função de transição*, descreve um passo arbitrário da máquina e satisfaz as restrições mencionadas acima. Suponha que a máquina esteja num estado q em $Q \setminus H$ e que sob a cabeça de leitura esteja o símbolo σ. Se $\delta(q, \sigma) = (q', \sigma', d)$, o próximo estado será q', o símbolo σ' será escrito no lugar de σ e a cabeça de leitura de M moverá de acordo com d.

A *configuração atual* de M é uma tripla $(w_1, q, w_2) \in \Sigma^* \times Q \times \Sigma^*$, onde w_1 é a palavra que aparece na fita à esquerda da cabeça de leitura, q é o estado atual e w_2 é a palavra à direita da cabeça de leitura ignorando-se brancos à direita. A cabeça de leitura aponta para a posição que contém o último símbolo de w_1. A configuração inicial é a tripla $(\triangleright, s, x)$, onde x é a entrada para a máquina.

Dizemos que uma configuração (w_1, q, w_2) *produz em k passos* uma configuração (w_1', q', w_2'), denotado por $(w_1, q, w_2) \vdash_M^k (w_1', q', w_2')$, se a máquina sai da primeira configuração e vai para a segunda em exatos k passos.

2.2. **Máquinas de Turing não-determinísticas.** A máquina de Turing não-determinística (MTND) é uma generalização da máquina de Turing determinística. Essa máquina tem um papel fundamental na teoria de complexidade pois está na base da definição da classe **NP**, conforme será mostrado posteriormente.

A maior parte das definições e idéias envolvidas na descrição das MTDs também se aplica às MTNDs. A diferença entre a MTND e a MTD está na função de transição e na maneira como as transições são feitas a cada passo. Nas máquinas determinísticas, existe uma única transição possível a partir de uma dupla (q, σ), onde q é um estado não-final e σ é um símbolo em Σ. Já nas máquinas não-determinísticas, pode existir mais de uma transição válida a partir de uma dupla (q, σ). Em cada passo da MTND, uma transição válida é escolhida arbitrariamente e é executada.

O número de transições válidas a partir de uma dupla (q, σ) é limitado superiormente por $3 \times |\Sigma| \times |Q|$. Logo, o número de configurações que podem ser produzidas a partir de cada configuração também é limitado superiormente por $3 \times |\Sigma| \times |Q|$. A transição numa MTND é portanto uma *relação* e não necessariamente uma função.

Formalmente, uma máquina de Turing não-determinística é uma quíntupla $M := (Q, \Sigma, \Delta, s, H)$, onde Q é o conjunto de estados, Σ é o alfabeto de símbolos, Δ é uma relação de $(Q \setminus H) \times \Sigma$ em $Q \times \Sigma \times \{\leftarrow, \downarrow, \rightarrow\}$, $s \in Q$ é o estado inicial e $H \subseteq Q$ é o conjunto de estados finais.

É evidente que as MTDs são casos particulares de MTNDs em que Δ é uma função, já que toda função é uma relação.

Uma transição (q', σ', d) em $\Delta(q, \sigma)$, onde $q \in Q$ e $\sigma \in \Sigma$, é interpretada como antes. Assim, q' será o próximo estado da máquina, σ' deve ser escrito no lugar de σ e d indicará o deslocamento da cabeça de leitura. Tal transição

será válida somente se $(q', \sigma', d) \in \Delta(q, \sigma)$. A definição de configuração é igual à que usamos na definição da máquina de Turing determinística. Para MTNDs, dizemos que uma configuração (w_1, q, w_2) produz a configuração (w_1', q', w_2') em k passos se, estando a máquina na primeira configuração, após k passos válidos ela pode estar na segunda a partir de uma seqüência de transições válidas.

A existência de várias transições possíveis para cada par (q, σ) numa MTND permite a ocorrência de computações distintas para uma mesma entrada. Como as MTNDs podem ter várias computações válidas possíveis para uma mesma entrada, elas podem produzir saídas diferentes para uma mesma entrada. Comentaremos mais sobre isso quando falarmos das linguagens decididas por uma máquina de Turing.

2.3. **Máquina de Turing probabilística.** Uma máquina de Turing probabilística (MTP) é uma MTND $M = (Q, \Sigma, \Delta, s, H)$ que funciona de maneira diferente.

A cada passo, em vez de uma transição ser escolhida arbitrariamente dentre todas as transições aplicáveis, escolhe-se uma com probabilidade uniforme. Assim, pode-se falar na probabilidade da MTP produzir, numa computação para uma certa entrada, uma saída específica. Todas as definições dadas para as MTNDs aplicam-se de maneira natural a uma MTP. Observe que uma MTD é uma MTP em que, a cada passo, há apenas uma transição aplicável.

Uma MTP corresponde à formalização do conceito de um computador acoplado a um gerador de números aleatórios.

É possível descrever o funcionamento de uma MTP postergando as escolhas aleatórias para o final. Ou seja, pode-se calcular a probabilidade de se estar em uma configuração específica a cada passo e só ao final fazer um único sorteio para determinar a configuração em que a máquina termina.

2.4. **Máquina de Turing quântica.** A definição de uma máquina de Turing quântica (MTQ) assemelha-se à de uma MTP. Uma máquina de Turing quântica é uma MTND $M = (Q, \Sigma, \Delta, s, H)$ com a relação Δ substituída por uma função

$$\alpha : Q \times \Sigma \times Q \times \Sigma \times \{\leftarrow, \downarrow, \rightarrow\} \longrightarrow \mathbb{C}$$

tal que, para cada (q, σ) em $Q \times \Sigma$, vale que

$$\sum_{(q', \sigma', d) \in T} |\alpha(q, \sigma, q', \sigma', d)|^2 = 1,$$

onde $T := Q \times \Sigma \times \{\leftarrow, \downarrow, \rightarrow\}$.

Cada número $\alpha(q_1, \sigma_1, q, \sigma, d)$ é chamado de *amplitude*. Note que α determina uma distribuição de probabilidade nas possíveis transições aplicáveis a um par (q_1, σ_1).

Define-se *superposição de configurações* como uma combinação linear de configurações, onde o coeficiente de uma configuração c é um número complexo α_c e $\sum_c |\alpha_c|^2 = 1$, sendo que o somatório é sobre todas as configurações c que estão na superposição. O coeficiente α_c é a *amplitude* da configuração c.

A primeira diferença no funcionamento de uma MTQ frente às máquinas anteriores é que esta, a cada instante, encontra-se numa superposição de configurações. Para que a transição de uma MTQ esteja bem definida, é preciso que a máquina satisfaça a seguinte propriedade: se, numa superposição obtida depois de k passos a partir da configuração inicial, há uma configuração com amplitude não-nula num estado final, então todas as configurações com amplitude não-nula nessa superposição estão num estado final.

Se todas as configurações da MTQ com amplitude não-nula forem finais, a MTQ termina a execução. Caso contrário, ela efetua uma transição, que consiste no seguinte. Digamos que $\psi := \sum_{j=1}^{t} \alpha_j c_j$ é a superposição corrente, onde $\sum_{j=1}^{t} |\alpha_j|^2 = 1$ e $\alpha_j \neq 0$ para todo j. Para cada j, seja C_j o conjunto das configurações que podem ser obtidas de c_j em um passo por meio de uma transição cuja amplitude é não-nula. Para cada c em C_j, seja α_c^j a amplitude correspondente à transição de c_j para c. Então, temos que a superposição resultante da transição é $\psi' = \sum_{j=1}^{t} \alpha_j \sum_{c \in C_j} \alpha_c^j c$.

Quando uma mesma configuração c aparece em mais de um conjunto C_j e $\left|\sum_{j:c \in C_j} \alpha_j \alpha_c^j\right|^2 \neq \sum_{j:c \in C_j} \left|\alpha_j \alpha_c^j\right|^2$, dizemos que houve *interferência*. A interferência é *negativa* se o lado esquerdo é menor que o direito e *positiva* caso contrário. O fenômeno de interferência é o principal responsável pela diferença entre uma MTQ e uma MTP.

A segunda diferença no funcionamento de uma MTQ frente às máquinas anteriores é que, ao final de sua execução, a MTQ efetua o que chamamos de *medição*. Digamos que $\psi := \sum_{j=1}^{t} \alpha_j c_j$ é a superposição final da máquina, com $\sum_{j=1}^{t} |\alpha_j|^2 = 1$ e $\alpha_j \neq 0$ para todo j. Uma medição consiste na escolha aleatória de uma configuração $c = c_j$ com probabilidade $|\alpha_j|^2$, e na transição da superposição ψ para a superposição $\psi' := c$, ou seja, para a superposição em que apenas a configuração c tem amplitude não-nula (mais exatamente, c tem amplitude 1 em ψ'). A saída da MTQ é o que está escrito na fita após a medição. Assim como uma MTP, uma MTQ pode produzir diferentes saídas para uma mesma entrada, cada uma com uma probabilidade.

Chamamos de *paralelismo quântico* à capacidade da MTQ estar numa superposição de configurações, e, num passo, efetuar transições múltiplas, envolvendo diversas configurações. Potencialmente é possível explorar o fenômeno de interferência bem como o paralelismo quântico para se obter algoritmos quânticos eficientes para problemas considerados difíceis no modelo clássico de computação.

3. Teoria clássica de complexidade

Apresentamos alguns resultados e definições do modelo clássico de computação com o intuito de estabelecer um paralelo durante a apresentação dos correspondentes quânticos desses resultados e definições.

3.1. **Principais classes de complexidade.** Vamos falar agora sobre as classes de complexidade mais importantes no modelo clássico de computação, baseados na abordagem apresentada por Papadimitriou [Pap94].

Para isso, definiremos o significado de tempo e espaço consumido nas máquinas do modelo clássico. Em seguida, falaremos das linguagens decididas por cada uma dessas máquinas e finalmente definiremos as principais classes do modelo clássico de computação.

Tempo consumido pelas máquinas de Turing. Durante a descrição da MTD, foi definido que $(w_1, q, w_2) \vdash_M^k (w_1', q', w_2')$ se a máquina M sai da primeira configuração e vai para a segunda em exatos k passos. Se (w_1, q, w_2) é a configuração inicial de M e q' é um de seus estados finais, então dizemos que k é o tempo consumido por M para a entrada w_2.

Dada uma MTD M, se existe um polinômio $p(n) : \mathbb{N} \longrightarrow \mathbb{N}$ tal que, para qualquer entrada x, o tempo consumido por M é limitado superiormente por $p(|x|)$, onde $|x|$ denota o comprimento da palavra x, então dizemos que M é *polinomialmente limitada.*

Se M é uma MTND, o maior tempo consumido em uma computação válida de M para uma entrada determinada é considerado o tempo consumido por M para essa entrada. Analogamente, M é polinomialmente limitada se existe um polinômio $p(n) : \mathbb{N} \longrightarrow \mathbb{N}$ tal que, para qualquer entrada x, o tempo consumido por M em qualquer computação válida é limitado superiormente por $p(|x|)$.

Por fim, se M é uma MTP, valem as mesmas definições usadas para as MTNDs. Vale destacar que no caso das MTPs também aplica-se o conceito de tempo *esperado* de computação, que formalmente é a esperança do tempo consumido por uma computação de M para uma dada entrada x.

Espaço consumido pelas máquinas de Turing. Existem também classes de problemas que são definidas em função do espaço consumido pelas máquinas de Turing que os resolvem. Da mesma forma que no estudo do consumo de tempo, temos interesse especial nos problemas que podem ser resolvidos por máquinas de Turing que consomem espaço polinomial no tamanho das entradas.

As diferenças entre as máquinas de Turing de naturezas distintas não são muito grandes no consumo de espaço. Mais adiante, ao enunciarmos um resultado de Savitch [Sav70], ficará mais claro o porquê dessa afirmação.

Dizemos que o *espaço consumido* por uma MT é a maior quantidade de células distintas usadas pela máquina para realizar uma computação válida para uma determinada entrada. No caso das MTDs, é o número de células

usadas na única computação possível. Já para MTNDs e MTPs, é o maior número de células utilizadas em uma computação válida.

Como em toda MT a cabeça de leitura só pode se deslocar de uma célula a cada passo, claramente o espaço consumido em uma computação é limitado superiormente pelo número de passos utilizados pela máquina.

Dizemos que uma MT *consome espaço polinomial* se o espaço consumido pela MT é limitado superiormente por um polinômio definido em função do tamanho da entrada.

Linguagens e máquinas de Turing. Ao definir as máquinas de Turing, utilizamos como parâmetro um alfabeto Σ. Esse alfabeto é um conjunto que contém todos os símbolos reconhecidos pela máquina de Turing em questão. Logo, uma entrada válida para a máquina deve ser uma palavra composta apenas por símbolos de Σ.

Um conjunto de palavras cujas letras pertencem a um alfabeto Σ é chamado de *linguagem*. Algumas linguagens possuem propriedades que as tornam especialmente interessantes. Geralmente estamos interessados em verificar se uma determinada palavra pertence ou não a uma particular linguagem L.

Máquinas de Turing que devolvem respostas binárias podem ser usadas para efetuar essa tarefa. Uma das respostas indica que a palavra fornecida como entrada pertence à linguagem L (aceitação) e a outra que a palavra não pertence (rejeição).

Em muitos casos, porém, deseja-se fornecer uma entrada para uma máquina de Turing para a obtenção de uma saída que não pode ser descrita apenas com duas respostas distintas. Por exemplo, uma máquina que recebe a representação binária de um número inteiro x qualquer como entrada e devolve como resposta a representação binária do número $x + 2$ claramente deve ser capaz de devolver mais do que duas respostas distintas. Nesse caso, dizemos que estamos lidando com um *problema*, e não com uma linguagem. Rigorosamente, existem diferenças entre problemas e linguagens, mas como podemos transformar uma coisa na outra de maneira razoavelmente simples, vamos falar de problemas e linguagens de maneira indistinta. Mostraremos agora as relações entre as linguagens e as máquinas de Turing determinísticas, não-determinísticas e probabilísticas.

Se existe uma MTD M que, dada uma palavra x, responde se x pertence ou não a uma determinada linguagem L, então dizemos que M *decide* a linguagem L. As respostas são devolvidas por M através dos símbolos 0 e 1, que indicam rejeição e aceitação, respectivamente, da palavra x. Esses símbolos devem aparecer sozinhos na fita da máquina após a mesma ter parado, na segunda célula da esquerda para a direita.

No caso das MTNDs, já vimos que podem existir várias computações e várias saídas possíveis para uma máquina e uma entrada. Assim, dizemos que uma MTND M decide uma linguagem L se toda palavra $x \in L$ é aceita por M em alguma de suas computações, e se toda palavra $x \notin L$ é rejeitada

por M em todas as suas computações. Essa definição acaba mostrando a característica das MTNDs que faz com que elas pareçam ser mais eficientes computacionalmente e menos realistas do que as MTDs.

Por fim, a decisão de linguagens por MTPs têm um aspecto um pouco diferente das demais máquinas de Turing. Assim como as MTNDs, as MTPs também podem produzir respostas distintas para uma mesma entrada. Porém, nas MTPs, atribuímos a cada computação válida (e conseqüentemente a cada possível resposta) uma determinada probabilidade. A decisão de uma linguagem por uma MTP envolve as probabilidades de obtenção das respostas.

Em alguns casos estamos interessados em fixar uma probabilidade máxima para as rejeições incorretas, em outros para as aceitações incorretas e em outros para as duas simultaneamente. A decisão de linguagens em MTPs, portanto, não possui uma definição única como no caso das MTDs e das MTNDs. Logo, ao utilizar esse conceito mais adiante no texto, definiremos exatamente o significado adequado dentro do contexto.

As classes **P** *e* **PSPACE**. A partir do conceito de máquinas de Turing polinomialmente limitadas, vamos definir as linguagens *polinomialmente decidíveis*. Dizemos que uma linguagem é polinomialmente decidível se existe uma máquina de Turing determinística polinomialmente limitada que a decide.

A classe **P** (*polynomial-time*) é o conjunto de todas as linguagens polinomialmente decidíveis. Esta classe é extremamente importante, pois contém a maior parte dos problemas que podem ser resolvidos de maneira eficiente. Dizemos que um problema é resolvido de maneira eficiente se existe um algoritmo para resolvê-lo que consome tempo polinomial no tamanho da entrada. A classe **P** é fechada sob união, intersecção, concatenação e estrela de Kleene. Essas propriedades são importantes e suas provas são interessantes, mas serão omitidas nesse texto.

De maneira bastante parecida com a que definimos a classe **P**, podemos definir a classe **PSPACE** (*polynomial-space*). Vale ressaltar que, enquanto a primeira faz a classificação das linguagens em função do tempo consumido, a segunda o faz em função do espaço consumido. Dizemos que uma linguagem pertence à classe **PSPACE** se ela é decidida por uma máquina de Turing determinística que consome espaço polinomial no tamanho da entrada.

É fácil ver que $\mathbf{P} \subseteq \mathbf{PSPACE}$: como já observamos, toda MTD que consome tempo polinomial certamente consome espaço polinomial, já que, a cada passo, a cabeça de leitura da máquina só pode se mover de no máximo uma célula à direita. Por outro lado, o inverso não é verdade. É fácil imaginar uma MTD que consome tempo superpolinomial mas espaço polinomial. Assim, é concebível (e até de se esperar) que existam linguagens em **PSPACE** que não estejam em **P**. Surpreendentemente, não se sabe até hoje se este é o caso ou não. Ou seja, não se sabe se $\mathbf{P} = \mathbf{PSPACE}$.

As classes **NP** *e* **coNP**. As classes **NP** e **coNP** podem ser descritas de maneira parecida com a que descrevemos a classe **P**. Conforme explicaremos adiante, essas classes são muito importantes no estudo da teoria de complexidade clássica.

Vamos definir **NP** (*nondeterministic polynomial-time*) e **coNP** em função das máquinas de Turing não-determinísticas polinomialmente limitadas. Dizemos que uma linguagem pertence à classe **NP** se ela pode ser decidida por uma máquina de Turing não-determinística polinomialmente limitada.

A classe das linguagens cujo complemento pertence a **NP** é denominada **coNP**, onde o complemento de uma linguagem L sob um alfabeto Σ é a linguagem $\Sigma^* \setminus L$. De maneira geral, o complemento de uma classe de complexidade arbitrária **C** é denotado por **coC** e definido como o conjunto das linguagens cujo complemento está em **C**.

Por exemplo, ao definir a classe **P**, também poderíamos ter definido a classe **coP**, seguindo a definição acima. Nesse caso, temos uma classe que é igual ao seu complemento. Dada uma linguagem L em **P** ou em **coP**, e uma máquina M polinomialmente limitada que a decide, basta trocar aceitação por rejeição e vice-versa na descrição de M para obter uma máquina polinomialmente limitada que decide o complemento de L.

As classes **NP** e **coNP** contêm vários problemas para os quais não se conhece algoritmo polinomial, e formam com a classe **P** a fronteira entre o que pode e o que não pode ser resolvido eficientemente no modelo clássico. Essas classes também são importantes porque contêm uma grande quantidade de problemas de interesse prático.

Claramente, toda linguagem que está em **P** também está em **NP** e em **coNP**, visto que uma MTD é uma MTND. Porém, não sabemos muito mais a respeito da relação entre essas três classes. A questão mais importante e conhecida é se **P** é igual a **NP**. Se isso for verdade, saberemos que muitos problemas para os quais hoje não se conhecem algoritmos exatos razoáveis terão uma solução polinomial. Porém, são poucos os que acreditam nessa hipótese.

NP $\subseteq$ **PSPACE**. Uma relação conhecida entre classes de complexidade é que **NP** $\subseteq$ **PSPACE**. Isso significa que toda linguagem decidida por uma MTND limitada polinomialmente pode ser decidida por uma MTD que consome espaço polinomial.

Vamos apresentar a prova de maneira razoavelmente informal, pois o que queremos mostrar é a idéia que está por trás da simulação de uma MTND por uma MTD.

Dada uma MTND M polinomialmente limitada que decide uma linguagem L, queremos mostrar que essa máquina pode ser simulada por uma MTD M' que consome espaço polinomial e decide a mesma linguagem L.

A diferença entre as máquinas M e M' é o não-determinismo. Sendo uma MTND, M requer apenas que uma de suas possíveis computações aceite

uma palavra da linguagem L, e também requer que todas as computações rejeitem as palavras que não estão em L.

Logo, ao simular M, a máquina M' deve ser capaz de simular todas as computações possíveis de M para poder rejeitar uma palavra corretamente. A conclusão de que uma palavra está na linguagem pode ser rápida (basta encontrar a primeira computação que aceita a palavra de entrada), mas de qualquer forma em geral não é possível determinar a priori quantas simulações precisam ser feitas.

Assim, considerando o pior caso, é fácil ver que toda simulação de M por M' que consiste em testar cada uma das computações possíveis pode consumir tempo exponencial no tamanho da entrada. Porém, sabemos que M é polinomialmente limitada. Logo, todas as suas computações consomem tempo, e portanto espaço, polinomial no tamanho da entrada.

Assim, uma simulação de M por M' que testa cada computação possível consumindo espaço polinomial e que sempre consegue reaproveitar esse espaço nas próximas computações é suficiente para mostrar que **NP** está contida em **PSPACE**.

A construção da M' é razoavelmente simples. A idéia principal consiste numa espécie de busca em largura num grafo. Mais especificamente, M' simula todas as computações possíveis de M com t passos antes de simular qualquer computação com $t+1$ passos.

Para representar as computações, M' pode indexar as transições das computações com números. Esses números serão obtidos a partir de um contador, que será representado na base binária. Logo, mesmo se o número de computações for exponencial, o espaço consumido por M' para armazenar tal contador será polinomial no tamanho da entrada.

A cada passo da simulação, M' incrementa seu contador e simula a computação de M referente a seu valor. O conteúdo da fita de M para cada computação pode ser guardado numa fita auxiliar durante a simulação, e antes da próxima computação essa fita é apagada. Dessa forma, é claro que o espaço usado para guardar as computações também será polinomial no tamanho da entrada.

Como toda transição de M pode ser simulada em tempo polinomial por M', a simulação pode ser feita consumindo espaço polinomial no tamanho da entrada. Assim, $\mathbf{NP} \subseteq \mathbf{PSPACE}$. Analogamente, mostra-se que $\mathbf{coNP} \subseteq \mathbf{PSPACE}$.

Dizemos que uma linguagem pertence à classe **NPSPACE** se ela é decidida por uma máquina de Turing não-determinística que consome espaço polinomial no tamanho da entrada. Uma simulação semelhante a essa foi proposta por Savitch [Sav70] para mostrar que $\mathbf{NPSPACE} = \mathbf{PSPACE}$.

As classes **RP** *e* **coRP**. As classes discutidas a seguir envolvem computações probabilísticas e são de fundamental importância tanto no modelo clássico como no modelo quântico.

Primeiro definimos a classe **RP** (*randomized polynomial-time*). Na literatura, essa classe é definida em função das *máquinas de Turing Monte-Carlo*. Infelizmente não há um consenso quanto a definição dessas máquinas (algumas fontes falam que elas devem ser limitadas polinomialmente, enquanto outras não estabelecem essa restrição). Logo, vamos fazer a definição da classe em função de suas propriedades principais.

Para que uma linguagem L pertença à classe **RP**, deve existir uma MTP M limitada polinomialmente satisfazendo o seguinte:

(1) Se uma palavra x dada na entrada não está na linguagem L, então a máquina M sempre rejeita a palavra x.
(2) Se uma palavra x dada na entrada está na linguagem L, então a máquina M aceita x com probabilidade pelo menos 0,5.

Essas duas propriedades caracterizam a máquina de Turing Monte-Carlo. Diz-se neste caso que M decide a linguagem L com probabilidade de aceitação 0,5.

O valor da probabilidade de aceitação na definição de **RP** não é muito importante, no sentido de que qualquer outro valor em $(0; 1]$ leva à mesma classe **RP**. Isso porque, de uma MTP limitada polinomialmente que decide uma linguagem L com probabilidade de aceitação p, para um valor p qualquer em $(0, 1]$, é possível obter uma MTP limitada polinomialmente que decide L com probabilidade de aceitação 0,5. De fato, se $p \geq 0{,}5$, não há nada a fazer. Se $p < 0{,}5$, executa-se k vezes, com $k = \lceil -1/\log p \rceil$, a máquina original e rejeita-se a palavra se pelo menos uma vez a máquina original a rejeitou. Do contrário, aceita-se a palavra. Esse procedimento aumenta a probabilidade de aceitação para pelo menos 0,5.

O uso da repetição da execução das máquinas de Turing Monte-Carlo é bastante útil, pois pode deixar a probabilidade de falsas rejeições arbitrariamente pequena. Por isso, esse recurso é bastante utilizado na prática, mesmo quando as probabilidades de rejeição incorreta são pequenas (menores que 0,5).

A classe **coRP**, das linguagens cujo complemento está em **RP**, pode também ser vista como a classe das linguagens L para as quais existe uma MTP M limitada polinomialmente com as seguintes propriedades:

(1) Se uma palavra x dada na entrada não está na linguagem L, então a máquina M rejeita x com probabilidade maior ou igual a 0,5.
(2) Se uma palavra x dada na entrada está na linguagem L, então a máquina M nunca rejeita a palavra x.

As definições deixam claro o caráter complementar das duas classes. Enquanto as linguagens que estão em **RP** admitem máquinas que fazem rejeições incorretas, as linguagens de **coRP** admitem máquinas que fazem aceitações incorretas.

A classe **RP** claramente está contida em **NP**. Basta notar que uma máquina de Turing Monte-Carlo para uma linguagem L é uma máquina de Turing não-determinística que decide L (existe pelo menos uma computação

dessa máquina que aceita cada palavra que está em L e todas as computações rejeitam palavras que não estão em L). Um argumento análogo mostra que a classe **coRP** está contida na classe **coNP**.

Também é claro que **P** está contida nessas duas classes, pois uma MTD é um caso especial de uma máquina de Turing Monte-Carlo, onde as palavras sempre são aceitas e rejeitadas corretamente.

As classes **ZPP** *e* **BPP**. Na última seção, vimos classes de linguagens que são decididas por MTPs limitadas polinomialmente que podem ou aceitar ou rejeitar erroneamente uma determinada palavra.

Agora, vamos definir uma classe de linguagens que exige das máquinas que as respostas sejam corretas, mas que abre mão da certeza da polinomialidade na execução.

Para que uma linguagem L pertença à classe **ZPP** (*zero-error probability polynomial-time*), deve existir uma máquina de Turing probabilística M que sempre aceita palavras que pertencem a L e sempre rejeita palavras que não pertencem a L, e cujo tempo esperado de execução para qualquer entrada é polinomial.

Pela definição, podemos perceber que a classe **ZPP** é muito parecida com a classe **P**, diferindo apenas no fato de que as máquinas utilizadas podem consumir tempo superpolinomial em algumas computações.

Outra definição possível é a seguinte: **ZPP** é a classe das linguagens que pertencem tanto a **RP** como a **coRP**. Se uma linguagem está tanto em **RP** como em **coRP**, executa-se de maneira alternada cada uma das máquinas envolvidas até que uma delas obtenha uma resposta certamente correta. O número de passos desse algoritmo é indeterminado (não há sequer garantias de que ele pára em algum momento), mas a esperança do número de passos é um valor polinomial no tamanho da entrada, pois a probabilidade de obtenção de respostas erradas diminui exponencialmente a cada execução das máquinas.

Outra classe de complexidade importante, que também pode ser relacionada com as demais classes probabilísticas já citadas, é a classe **BPP** (*bounded-error probabilistic polynomial-time*), que contém as linguagens para as quais existem máquinas de Turing que devolvem respostas erradas (tanto rejeições como aceitações) com probabilidade estritamente menor que 0,5. Na verdade, qualquer delimitação da probabilidade no intervalo $(0; 0{,}5)$ resultaria na mesma classe. Porém, Papadimitriou [Pap94] mostrou que delimitações maiores ou iguais a 0,5 podem levar a uma classe diferente.

A definição dessa classe é simétrica, pois rejeição e aceitação são feitas corretamente na maioria absoluta das computações feitas por essas máquinas. Assim, utilizando um argumento análogo ao do resultado $\mathbf{P} = \mathbf{coP}$, obtemos que $\mathbf{BPP} = \mathbf{coBPP}$.

Além disso, é fácil ver que $\mathbf{RP} \subseteq \mathbf{BPP}$. Dadas uma linguagem L em **RP** e uma máquina M associada, basta executar M com a entrada desejada duas vezes para que a probabilidade de falsa rejeição seja de no máximo 0,25.

Como a probabilidade de falsa aceitação é zero, as condições necessárias para que L pertença a **BPP** estão satisfeitas. O argumento que mostra que **coRP** $\subseteq$ **BPP** é análogo.

Por fim, vale destacar que não sabemos ainda se a classe **BPP** está contida na classe **NP**.

4. Classes quânticas de complexidade

Vamos agora apresentar as duas principais classes quânticas de complexidade e demonstrar alguns resultados que as relacionam às classes clássicas.

4.1. Tempo, espaço e linguagens no modelo quântico.

Durante as discussões sobre o modelo clássico de computação, fizemos considerações sobre o consumo de tempo e de espaço pelas máquinas de Turing determinísticas, não-determinísticas e probabilísticas.

O tempo consumido por uma MTQ com entrada x é o número de passos que a máquina efetua até terminar a execução. Essa definição é consistente, porque exigimos que, quando uma configuração da superposição atinge um estado final, todas as demais configurações também o fazem.

O espaço consumido por uma MTQ M com entrada x é definido de maneira análoga ao espaço consumido por uma MTND. Dentre todas as configurações que durante a execução de M tiveram amplitude não-nula, seja c aquela com o maior número possível de células em que M escreveu algo. O número de células utilizadas em c é o espaço consumido por M com a entrada x.

No contexto de linguagens, estamos interessados em MTQs que, para cada entrada x, têm saída ou $x1$ ou $x0$ (MTQs devem ser reversíveis, por isso a resposta usualmente binária é precedida pela entrada). Diz-se que uma MTQ M desse tipo decide de maneira exata uma linguagem L se, para todo x em L, a máquina M produz como saída $x1$, e, para todo x fora de L, a máquina M tem como saída $x0$.

Por fim, para um número p entre 0 e 1, diz-se que uma MTQ M decide L com probabilidade p se M, com entrada x em L, produz $x1$ com probabilidade pelo menos p e, com entrada x fora de L, produz $x0$ com probabilidade pelo menos p.

4.2. As classes EQP e BQP.

As duas classes de complexidade que iremos introduzir aqui foram propostas por Bernstein e Vazirani [BV97].

A classe **EQP** é o conjunto das linguagens que são decididas de maneira exata por uma MTQ que consome tempo polinomial no tamanho da entrada. Essa classe corresponde à classe **P** do modelo clássico.

A classe **BQP** é o conjunto das linguagens para as quais existem máquinas de Turing quânticas que devolvem respostas erradas (tanto rejeições como aceitações) com probabilidade estritamente menor que 0,5. A classe correspondente no modelo clássico é **BPP**, e assim como no caso dela, a probabilidade de erro pode ser qualquer valor no intervalo $(0; 0{,}5)$.

A seguir, vamos mostrar algumas relações envolvendo essas classes e as classes tradicionais de complexidade.

4.3. **Relações envolvendo as classes quânticas.** Inicialmente, vamos mostrar que $\mathbf{P} \subseteq \mathbf{EQP}$ e que $\mathbf{BPP} \subseteq \mathbf{BQP}$, pois as demonstrações e as idéias envolvidas são mais simples. Em seguida, mostraremos que $\mathbf{BQP} \subseteq \mathbf{PSPACE}$. Seguem abaixo demonstrações breves das duas primeiras relações, apresentadas no artigo de Bernstein e Vazirani [BV97].

Teorema 4.1. $\mathbf{P} \subseteq \mathbf{EQP}$.

Demonstração. Seja L uma linguagem que pertence à classe **P**. É fácil ver que existe uma MTD limitada polinomialmente que, para cada entrada x, produz como saída $x1$ se $x \in L$ e $x0$ caso contrário. Para toda MTD polinomialmente limitada, existe uma MTD reversível equivalente, que também é limitada polinomialmente. Para obtê-la, modifica-se a máquina original de modo que cada um de seus estados só possa ser atingido por movimentos da cabeça de leitura em uma única direção (as transições cujas triplas têm o mesmo estado devem fazer o mesmo deslocamento da cabeça de leitura da máquina).

Isso pode ser obtido duplicando-se os estados da máquina. Com isso, a função de transição torna-se bijetora quando a direção é ignorada, e pode-se, de uma configuração arbitrária da máquina, deduzir-se a configuração "anterior". Ou seja, a máquina é reversível. Além disso, o número de passos executados pela máquina até que ela pare é o mesmo da máquina original.

Mas se uma MTD é reversível, então essa máquina é uma MTQ onde cada superposição de configurações contém apenas uma configuração com amplitude não-nula.

Logo, existe uma MTQ limitada polinomialmente que, para cada entrada x, devolve $x1$ como resposta sempre que x está em L, e devolve $x0$ sempre que x está fora de L. Então, L é decidida de maneira exata por uma MTQ, e portanto pertence a **EQP**. Concluímos assim que $\mathbf{P} \subseteq \mathbf{EQP}$. □

A demonstração a seguir mostra como uma MTQ engloba naturalmente um gerador de números aleatórios.

Teorema 4.2. $\mathbf{BPP} \subseteq \mathbf{BQP}$.

Demonstração. Sejam L uma linguagem em **BPP** e M uma MTP para L dada pela definição de **BPP**. Para mostrar que $L \in \mathbf{BQP}$, vamos descrever uma MTQ que simula M com um aumento apenas polinomial no consumo de tempo.

Para simplificar, vamos assumir que M tem k opções de transição em cada um de seus passos, para algum k fixo. Se numa simulação de M por uma MTQ M' conseguimos, a cada passo, escolher aleatoriamente um símbolo do alfabeto $\{1, \ldots, k\}$, então M pode ser simulada por M'. Cada símbolo do alfabeto é usado para definir qual transição se aplica no passo que está sendo simulado de M. Esses símbolos podem ser gerados da seguinte maneira.

A cada passo, se o estado atual não é final, a cabeça de leitura desloca-se para uma determinada posição da fita e escreve-se o símbolo j com amplitude $1/\sqrt{k}$, para $j = 1, \ldots, k$, em uma única transição, que representa um sorteio aleatório. Feito isso, o conteúdo da célula é lido, a cabeça retorna para a posição original da fita e a transição representada pelo símbolo lido é realizada.

A célula utilizada no sorteio deve ser tal que não haja interferência no resto da fita. Como M é polinomialmente limitada, existem números c e e tais que qualquer computação de M não consome mais do que cn^e passos. Assim, basta inserir os caracteres dos sorteios seqüencialmente, começando na (cn^{e+1})-ésima célula da fita.

Como cada transição do passo de sorteio tem amplitude $1/\sqrt{k}$, então cada configuração no final terá amplitude $(1/\sqrt{k})^t$, onde t é o número de passos de M, lembrando que não ocorrerá interferência entre as configurações geradas. Logo, com a medição, a probabilidade de obtenção de um determinado resultado em M' será igual à da obtenção do mesmo resultado em M.

Como M pode ser simulada por M', a linguagem L pode ser decidida com a mesma probabilidade $p > 0{,}5$ por uma MTQ limitada polinomialmente. Logo, $L \in \mathbf{BQP}$, e portanto temos que $\mathbf{BPP} \subseteq \mathbf{BQP}$. □

As duas provas mostradas acima servem para reforçar idéias que devem ser intuitivas após a leitura das definições envolvidas. Nosso objetivo agora é mostrar uma relação menos imediata e mais importante, que envolve o consumo de espaço na simulação de uma MTQ por uma MTD.

Quando falamos em simulação de uma MTQ por uma MTD, estamos nos referindo a uma simulação onde queremos saber a probabilidade de cada saída possível ser produzida. Não conhecemos nenhuma simulação de uma MTQ por uma MTD que consuma tempo polinomial no tempo consumido pela MTQ, porém o teorema abaixo mostra que é possível fazer uma simulação utilizando espaço polinomial no espaço consumido pela MTQ. Nesse texto, mostraremos apenas as linhas gerais da prova desse teorema.

Teorema 4.3. $\mathbf{BQP} \subseteq \mathbf{PSPACE}$.

Demonstração. Seja $L \in \mathbf{BQP}$ e $M := (Q, \Sigma, \alpha, s, H)$ uma MTQ polinomialmente limitada que decide L com probabilidade p maior que 0,5. Vamos descrever uma MTD M' que decide L em espaço polinomial no tempo consumido por M. A máquina M' calcula, para cada entrada x, a probabilidade p_x de M aceitar x (ou seja, de M produzir $x1$ como saída). No final, M' aceita ou rejeita x de acordo com o valor de p_x.

Dizemos que uma configuração (w_1, q, w_2) tem *tamanho* k se a cadeia de caracteres $w_1 w_2$ tem k caracteres. Dadas duas configurações c_1 e c_2, denotamos por $\alpha(c_1, c_2)$ o valor da amplitude correspondente à transição de M que leva c_1 a c_2. Se não há nenhuma transição que leva c_1 a c_2, então $\alpha(c_1, c_2) = 0$. Seja $c_0 := (\triangleright, s, x)$ a configuração inicial.

A máquina M', para cada inteiro t a partir de $|x|$, simula t passos de M e calcula a probabilidade p_x de M produzir, em t passos, a saída $x1$. Essa probabilidade é dada por

$$p_x = \left| \sum_{c_1,\ldots,c_t} \prod_i \alpha(c_{i-1}, c_i) \right|^2,$$

onde o somatório é sobre todas as configurações $c_1, \ldots, c_t$ de tamanho no máximo t onde $c_t = (\triangleright, h, x_1)$, para algum h em H. Note que o número de termos no somatório é no máximo $T = t|\Sigma|^{2t}|Q|$.

A máquina M' deve portanto calcular o somatório descrito acima. O primeiro problema que aparece é a incapacidade de M' fazer cálculos com números irracionais. MTDs só podem armazenar valores inteiros limitados, enquanto que cada $\alpha(c_{i-1}, c_i)$ pode nem mesmo ser racional. Logo, a simulação será uma aproximação, que deverá ter um erro tolerável. Por fim, vale destacar que cada um dos no máximo T termos do somatório terá t fatores.

Se cada $\alpha(c_{i-1}, c_i)$ for representado com m bits tanto na parte inteira como na parte imaginária dos números, onde m é grande comparado a $\log T$, o erro será pequeno (da ordem de 2^{-m}). Assim, a primeira dificuldade já foi eliminada.

O que M' deve fazer é computar cada termo do somatório e guardar a soma dos termos. Como cada operação de adição usa pouco espaço auxiliar (ou seja, consome espaço polinomial em m) e só é necessário guardar a cada operação a soma atual, podemos garantir que o espaço necessário para efetuar o somatório é polinomial. No caso dos produtos, os mesmos comentários se aplicam.

O que resta considerar agora é a obtenção de cada $\alpha(c_{i-1}, c_i)$. É fácil, em tempo polinomial nos comprimentos de c_{i-1} e c_i, detectar se c_i pode ou não ser derivada de c_{i-1} em um passo. Se não pode, então $\alpha(c_{i-1}, c_i) = 0$. Se pode, então ao mesmo tempo pode-se determinar a transição $(q_1, \sigma_1, q, \sigma, d)$ em $Q \times \Sigma \times Q \times \Sigma \times \{\leftarrow, \downarrow, \rightarrow\}$ que, quando aplicada a c_{i-1}, leva a c_i. A máquina M' então aciona uma "subrotina" que recebe um número m e devolve, em espaço polinomial em m, o valor de $\alpha(q_1, \sigma_1, q, \sigma, d)$ com precisão 2^{-m}. O valor devolvido pela subrotina é a aproximação desejada de $\alpha(c_{i-1}, c_i)$. Mostrar que de fato há uma MTD que efetua tal subrotina e consome espaço polinomial em m não é tarefa trivial e envolve uma série de etapas. Omitiremos essa prova nesse texto.

É importante notar que o número de tais subrotinas não depende da entrada, mas apenas da máquina M. Assim M' está bem definida, desde que existam MTDs que executem tais subrotinas. Mais do que isso, cada etapa que M' executa consome espaço polinomial (assumindo que as subrotinas consumam espaço polinomial), portanto de fato $\mathbf{BQP} \subseteq \mathbf{PSPACE}$. □

Dos resultados acima, temos que $\mathbf{BPP} \subseteq \mathbf{BQP} \subseteq \mathbf{PSPACE}$. Assim, não é possível mostrar que $\mathbf{BPP} \neq \mathbf{BQP}$ sem resolver a clássica questão de se $\mathbf{P}$ está propriamente contido em $\mathbf{PSPACE}$ ou não. Por outro lado, no mesmo

artigo, Bernstein e Vazirani [BV97] mostram um oráculo em relação ao qual $\mathbf{BPP} \neq \mathbf{BQP}$. Outros resultados nessa linha, porém direcionados à questão "$\mathbf{P} \neq \mathbf{NP}$?", foram provados por Bennet *et al.* [BBBV97]. Por exemplo, Bennet *et al.* mostraram que, em relação a um oráculo, $\mathbf{NP} \not\subseteq \mathbf{BQP}$.

5. Desafios e frustrações

A iniciação científica que deu origem a este trabalho de formatura começou na metade do ano passado. O tema foi escolhido porque a área de computação quântica era (e de certa forma continua sendo) razoavelmente misteriosa, e despertava grande interesse tanto em mim como no Marcel, com quem fiz essa iniciação. Sabíamos que o assunto era novo e que muitas coisas diferentes precisariam ser estudadas, mas mesmo assim achamos que ia valer a pena. No meu caso, o interesse principal sempre foi estudar a parte de complexidade computacional. Dessa forma, teria uma desculpa para estudar esse assunto no modelo clássico de computação também.

No início, tivemos que estudar conceitos básicos de teoria de complexidade computacional do modelo clássico, que envolviam basicamente as máquinas de Turing, e alguns conceitos de fisíca quântica. Naturalmente, a parte de computação foi muito mais fácil, pois tudo faz sentido e depois de um tempo fica intuitivo. Já a parte de física é muito complicada. Infelizmente, nunca tinha estudado nada de física quântica, e o começo foi bem duro. Nessa área, a intuição não ajuda em nada, e constantemente percebemos que coisas supostamente já entendidas podem estar completamente erradas.

Passada essa primeira etapa, começamos a estudar um pouco a parte de espaços de Hilbert. Basicamente, essa área envolve algebra linear com números complexos. Essa parte não foi muito difícil porque não nos aprofundamos muito, mas o assunto aparentemente é bastante denso.

Como o começo foi um pouco distante da computação, decidimos em seguida estudar um artigo de Rieffel e Polak [RP00] (recomendado pelo "coach" Aritanan Gruber) que dava uma visão geral da área. O artigo ajudou a retomar o ânimo e a reajustar algumas idéias que naquele momento estavam dispersas e desconexas em nossas cabeças.

Depois disso, decidimos estudar um artigo razoavelmente importante na área, que tratava principalmente de teoria de complexidade na computação quântica, da autoria de Bernstein e Vazirani [BV97]. Fiquei bastante animado no início, pois tratava do assunto que mais me interessava. Como estávamos em dúvida sobre como algumas coisas importantes funcionavam no modelo quântico (como medição de tempo, relação de máquinas de Turing com circuitos quânticos, etc), achamos que o artigo seria muito importante na compreensão do modelo.

Porém, passadas as primeiras semanas de leitura, percebemos que o artigo não estava bem escrito. Várias passagens eram obscuras, alguns teoremas tinham provas '"discutíveis" e alguns detalhes extras inseridos nas discussões

(principalmente os que envolviam precisão nas computações) complicaram bastante a compreensão do texto.

Mesmo assim, decidimos que algumas coisas importantes descritas no artigo precisavam ser escritas em nosso relatório. Foi esse o momento em que realmente comecei a escrever, e essa tarefa mostrou-se bem mais difícil do que eu esperava. Inicialmente tive muita dificuldade com o Latex, pois precisava usar vários símbolos e fórmulas matemáticas no texto (o que não tinha acontecido até então nos trabalhos feitos usando esse processador de texto).

Além disso, acabei descobrindo que escrever textos matemáticos razoáveis bons não é uma atividade trivial. Nesse momento a presença da orientadora foi indispensável, pois ela já tem experiência e sabe indentificar imprecisões e exposições imprecisas de idéias. No começo não gostava muito de reescrever a mesma coisa várias vezes, mas sempre tive consciência de que aquilo precisava ser feito.

Para poder escrever sobre a parte de complexidade, precisei aprofundar um pouco meus conhecimentos em teoria da complexidade. No período, cursei a matéria Algoritmos e Complexidade de Computação, que acabou me ajudando bastante na fixação de conceitos importantes. Com essa bagagem fui capaz de estudar tópicos que não foram abordados nesse curso, como classes probabilísticas, importantes no estabelecimento de paralelos entre o modelo clássico e o quântico de computação.

Na metade desse ano, logo após a entrega do primeiro relatório para a FAPESP, nossa orientadora decidiu que deveríamos submeter o trabalho que estávamos fazendo para a Jornada de Iniciação Científica do IMPA. Para isso, deveríamos separar o trabalho em dois: um contendo a parte de algoritmos quânticos, escrito pelo Marcel, e outro contendo teoria de complexidade, escrito por mim.

Nesse momento, nosso trabalho estava razoavelmente grande, mas algumas partes não estavam bem feitas. Muitas das partes escritas no início da iniciação estavam escritas numa linguagem inadequada e imprecisa. Além disso, essas partes eventualmente não estavam compatíveis com o que estava sendo escrito no momento. Por isso, foi necessário parar de escrever coisas novas para reorganizar o texto.

Essa atividade também foi bastante trabalhosa. Fazer com que um texto razoavelmente grande fique consistente é difícil, pois algumas notações são mais práticas em uns casos e outras são mais práticas em outros. Indiscutivelmente, foi bastante cansativo fazer tudo isso, mas inegavelmente a qualidade do texto aumentou.

Vale destacar também que a participação nas Jornadas arruinou nosso planejamento inicial. Gastamos quase um mês reorganizando o texto para submetê-lo a comissão julgadora, e como ele foi aceito precisamos reajustar tudo novamente para reenviar os dois trabalhos unidos em um só.

Na reta final, vamos organizar o texto da monografia do IMPA e o relatório que será enviado para a FAPESP. Ainda não fizemos tudo, mas certamente

isso consumirá um tempo enorme, pois o relatório da FAPESP terá por volta de 150 páginas e precisará ser revisto.

6. Disciplinas do BCC importantes para a iniciação científica

Conforme já foi adiantado na seção anterior, várias matérias do BCC foram importantes para o trabalho. Inicialmente, destaco Análise de Algoritmos e Princípio de Desenvolvimento de Algoritmos. Na minha opinião, essas matérias são dois marcos importantes no BCC, e certalmente são importantes em qualquer iniciação científica.

A parte de física necessária para essa iniciação não foi vista em nenhuma das duas matérias oferecidas no currículo obrigatório. Inicialmente achei isso ruim, mas quando vi os pré-requisitos de uma disciplina de Física Quântica no IF-USP, percebi que para o BCC seria inviável.

A teoria matemática que aprendemos no BCC também foi muito importante. Matérias como Algebra Linear e Métodos Numéricos de Algebra Linear, normalmente execrados por vários alunos do BCC, mostraram-se essenciais na iniciação. Às vezes, nós alunos reclamamos que os professores da Matemática não mostram a importância prática do que nos ensinam. Porém, percebi que fazer isso é muito difícil. Ferramentas como Cálculo e Álgebra Linear aparecem inesperadamente quando estamos estudando, e é simplesmente impossível para um professor mostrar algo que interesse a todos.

Durante os estudos da parte de Teoria da Computação (em especial, a parte de máquinas de Turing), surgiram matrizes estocásticas, que vemos no curso de Introdução aos Processos Estocásticos. Além disso, Estatística I também foi muito importante, pois as máquinas de Turing quânticas utilizam bastante os conceitos vistos na teoria de probabilidade.

Conforme já citei anteriormente, a matéria optativa Algoritmos e Complexidade de Computação foi bastante importante, pois ajudou a fixar conceitos essenciais no assunto. Consequentemente, a disciplina obrigatória Linguagens Formais e Autômatos também acabou sendo essencial. Em Análise de Algoritmos vimos um pouco de teoria de complexidada, mas na minha opinião isso é insuficiente. Pessoalmente, não acho razoável um aluno do BCC se formar sem saber o que é uma máquina de Turing.

As outras matérias da graduação não estavam intimamente ligadas com a iniciação científica, mas foram importantes para minha formação. Destaco principalmente Algoritmos em Grafos, pois gosto bastante desse assunto e porque os algoritmos que aprendemos foram bastante importantes nas maratonas de programação (também essenciais em minha formação).

As matérias da Matemática e da Estatítica também foram importantes. As álgebras são muito interessantes e nos ajudam a ver as coisas de uma outra maneira, e as matérias da Estatística mostram que a teoria pode ser usada diretamente em coisas bem práticas. Não utilizei muito o que aprendi em Cálculo, mas eventualmente precisei saber um pouco de integral e derivada. Ou seja, nem sempre encontramos aplicações diretas para o que vemos nessas

disciplinas, mas as idéias envolvidas sempre são úteis. Sinto arrependimento por não ter estudado mais essas matérias, pois acredito que muitas coisas teriam sido mais fáceis.

Outra matéira interessante foi a de Algoritmos de Aproximação. Essa área é razoavelmente recente, e muitos dos resultados vistos foram provados há poucos anos. É bem legal aprender algo que ainda está sendo desenvolvido (o que não acontece em matemática, onde estudamos basicamente assuntos do século XIX para trás).

Destaco também a matéria Tópicos de Matemática Discreta. Sem sombra de dúvidas, essa foi a matéria mais difícil que fiz na vida, e precisei estudar muito para passar. Porém, não me arrependo de ter cursado até o fim. Essa matéria certamente foi um divisor de águas para muitos que a cursaram. Para mim foi legal porque vi como assuntos extremamente interessantes podem usar praticamente tudo que vimos (ou deveríamos ter visto) de teoria no BCC, incluindo matemática.

Nesse ano cursei algumas matéirias como aluno especial de pós-graduação, e a experiência foi muito interessante. Introdução à Teoria dos Grafos foi muito legal, pois complementou a parte de teoria que não vemos na matéria de grafos da graduação. Ter feito a matéria com o professor Paulo Feofiloff foi muito bom, pois ele é exigente e nos força a fazer as demonstrações de maneira clara e sucinta (isso foi importante na iniciação também). Otimização Combinatória também está sendo bem legal (como já disse, gosto do assunto), e Biologia Computacional também é interessante por tratar de um assunto bastante recente.

Por fim, destaco a disciplina Desafios de Programação, que fiz no penúltimo semestre do curso. Acredito que essa matéria não deve ser vista apenas como preparação para maratonas, mas sim como uma boa revisão do curso. Nos problemas de maratona, temos que desenvolver soluções rapidamente e não temos um indicador prévio do algoritmo que será utilizado. Isso é muito diferente dos exercícios-programa que fazemos, pois nesses casos já sabemos o que deve ser feito. Além disso, os problemas de maratona precisam ser feitos em ambientes razoavelmente restritos. Ou seja, não temos à disposição várias bibliotecas com algoritmos prontos. Se quisermos usar algo mais complexo, precisamo fazer tudo na mão.

Na minha opinião, todo aluno formado do BCC deveria ser capaz de programar nessa situação. O uso de bibliotecas é importante em muitas situações, mas um cientista da computação deve saber implementar pelo menos alguns algoritmos importantes vistos no curso. Ao fazer essa matéria, acho que a pessoa descobre se realmente aprendeu alguma coisa, e também tem a chance de fazer uma excelente revisão de conteúdos centrais do BCC.

7. Ambiente de Trabalho

O IME proporciona um ambiente de trabalho muito bom para quem quer fazer iniciação científica ou qualquer atividade acadêmica. Vários professores tentam fazer o que podem para ajudar.

O tema da minha iniciação científica é razoavelmente novo, e é difícil encontrar material introdutório de qualidade sobre o assunto. Mesmo assim, tive apoio para estudar essa área. Essa nem é exatamente a área de pesquisa de minha orientadora, mas mesmo assim ela aceitou ajudar.

Foi muito bom também porque algumas coisas que estudamos eram obscuras para todos (não só para o Marcel e para mim, mas também para a própria orientadora), e essas situações de certa forma mostraram como as coisas podem se desenvolver mesmo quando não há a quem ou a que recorrer.

Outro fato interessante foi a ajuda de outros professores no decorrer da iniciação. Em alguns momentos, as dúvidas se concentravam em tópicos conhecidos mas pouco estudados por nós três. Nesses momentos, outros professores acabavam ajudando, e achei essas interações bem legal.

Por fim, o contato com o meio acadêmico também foi facilitado pela participação em algumas atividades extra-curriculares. Infelizmente não pude participar de vários seminários sugeridos pela orientadora, mas a participação em eventos como a maratona de programação e as Jornadas de Iniciação Científica do IMPA ajudou bastante.

O contentamento com o ambiente acadêmico me ajudou na decisão de continuar meus estudos, e pretendo fazer o mestrado em Ciência da Computação. Pretendo continuar estudando Computação Quântica, mas também quero me aprofundar um pouco mais na área de Complexidade Computacional.

8. Conclusão

Para concluir, gostaria de dizer que gostei muito do curso de Ciência da Computação oferecido no IME. Sem sombra de dúvidas, apredi muito nesses anos em que estive aqui, e acho que tive uma boa formação. Se não aprendi coisas importantes, a maior parte da culpa foi minha. Logo, gostaria de agradecer aos professores pelas aulas (muito não merecem, é verdade, mas vá lá).

Também gostaria de agradecer aos meus colegas de sala. Sem sombra de dúvidas, as brincadeiras de gosto duvidoso, as bagunças durante as aulas e as comunicações durante algumas provas foram muito importantes para minha sobrevivência no BCC. Sem sombra de dúvidas, fiz excelentes amizades aqui, e espero que todos tenham se divertido tanto quanto eu.

Por fim, gostaria de agradecer ao professor Carlos Eduardo Ferreira e ao "coach" do IME-USP, Aritanan Gruber (Pil) pelo estímulo à participação em maratonas de programação. Desde o primeiro ano o Carlinhos estimulou a participação em competições de programação, e acho que isso deveria ser feito por outros professores do departamento.

Com isso, acabei me interessando bastante, e acabei participando de maratonas de programação nos 4 anos de graduação. Nesse tempo, conheci programadores inacreditáveis, como o Pedro Eira, com quem o Marcel e eu fomos ao Havaí na final mundial da competiçao em 2002.

Sem sombra de dúvidas, aprendi muito com maratonas de programação. Aprender um assunto por vontade própria é muito diferente de aprender compulsoriamente. Matérias como Análise de Algoritmos e Algoritmos em Grafos foram tranqüilas para mim, e sem sombra de dúvidas devo isso às maratonas. Também aprendi muito com o Pil, com quem o Marcel e eu convivemos nesses 4 anos de competições. Aproveito para agradecer ao Marcel, companheiro de maratona desde o primeiro ano, e ao Pedro, ao Antônio e ao Guilherme, que fizeram parte de nossos times.

Referências

[AKS02a] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. Preprint. `http://www.cse.iitk.ac.in/news/primality.pdf`, 2002.

[AKS02b] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. Revised. `http://www.cse.iitk.ac.in/news/primality_v3.pdf`, 2002.

[BBBV97] C.H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani. Strengths and weaknesses of quantum computing. *SIAM J. Comput.*, 26(5):1510–1523, 1997.

[BV97] E. Bernstein and U. Vazirani. Quantum complexity theory. *SIAM J. Comput.*, 26(5):1411–1473, 1997.

[Chu33] A. Church. A set of postulates for the foundation of logic. *Annals of Mathematics*, 25:839–864, 1933.

[Chu36] A. Church. An unsolvable problem of elementary number theory. *Annals of Mathematics*, 58:345–363, 1936.

[Cob65] A. Cobham. The intrinsic computational difficult of functions. In Y. Bar-Hillel, editor, *Logic, Methodology and Philosophy of Science*, pages 24–30. North-Holland, 1965.

[Coo71] S. Cook. The complexity of theorem proving procedures. In *Proc. 3rd ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[Deu85] D. Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proc. Roy. Soc. London Ser. A*, 400(1818):97–117, 1985.

[Deu89] D. Deutsch. Quantum computational networks. *Proc. Roy. Soc. London Ser. A*, 425(1868):73–90, 1989.

[Edm65] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[Fey82] R. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6 & 7):467–488, 1982.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability: a Guide to the Theory of NP-Completeness*. Freeman, 1979.

[Göd31] K. Gödel. On formally undecidable propositions of Principia Mathematica and related systems. *Monatshefte fur Math. und Physik*, 38:173–198, 1931.

[Kar72] R.M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., 1972)*, pages 85–103. Plenum, New York, 1972.

[Kle36] S. Kleene. General recursive functions of natural numbers. *Mathematische Annalen*, 112:727–742, 1936.

[Kle52] S. Kleene. *Introduction to Metamathematics*. D. Van Nostrand, Princeton, NJ, 1952.

[Lev73] L.A. Levin. Universal sorting problems. *Problems of Information Transmission*, 9:265–266, 1973.

[Pap94] C.H. Papadimitriou. *Computational Complexity*. Addison-Wesley Publishing Company, Reading, MA, 1994.

[Pos36] E. Post. Finite combinatory process. *Journal of Symbolic Logic*, 1:103–105, 1936.

[RP00] E. Rieffel and W. Polak. Introduction to quantum computing. *ACM Computing Surveys*, 32(3):300–335, 2000.

[RSA78] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Comm. ACM*, 21(2):120–126, 1978.

[Sav70] W.J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

[Sho94] P.W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science (Santa Fe, NM, 1994)*, pages 124–134. IEEE Comput. Soc. Press, Los Alamitos, CA, 1994.

[Sho97] P.W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997.

[Tur36] A. Turing. On computable numbers with an application to the entscheidungsproblem. *Proc. London Math. Soc.*, 2:230–265, 1936.

[Tur37] A. Turing. Rectifications to 'on computable numbers...'. In *Proc. London Mathematical Society*, volume 4, pages 544–546, 1937.

Instituto de Matemática e Estatística, Universidade de São Paulo, Rua do Matão 1010, 05508–900 São Paulo, SP

Endereços Eletrônicos: {cardonha,cris}@ime.usp.br

URL: http://www.ime.usp.br/~magal/quantum