Maurício Machado Barbosa

Relações entre classes de problemas computacionais

Um estudo sobre classes de complexidade quânticas

Florianópolis - SC, Brazil

2020

Maurício Machado Barbosa

Relações entre classes de problemas computacionais

Um estudo sobre classes de complexidade quânticas

Monografia submetida ao Programa de Graduação em Ciências da Computação para a obtenção do Grau de Bacharel em Ciências da Computação.

Universidade Federal de Santa Catarina

Departamento de Informática e Estatística

Ciências da Computação

Orientadora: Prof[a]. Dra. Jerusa Marchi

Florianópolis - SC, Brazil

2020

Relações entre classes de problemas computacionais
Um estudo sobre classes de complexidade quânticas / Maurício Machado Barbosa. – Florianópolis - SC, Brazil, 2020- 67 p. : il.(alguma color.); 30 cm.
Orientadora:Profª. Dra. Jerusa Marchi

Trabalho de Conclusão de Curso de Graduação – Universidade Federal de Santa Catarina
Departamento de Informática e Estatística
Ciências da Computação , 2020.
1. Computação Quântica. 2. Classes de Complexidade. 3. Bounded Error Quantum Polynomial Time. I. Profª. Dra. Jerusa Marchi. II. Universidade Federal de Santa Catarina. III. Faculdade de Ciências da Computação. IV. Relações entre classes de problemas computacionais
Um estudo sobre classes de complexidade quânticas
CDU 02:141:005.7

Maurício Machado Barbosa

Relações entre classes de problemas computacionais
Um estudo sobre classes de complexidade quânticas

Monografia submetida ao Programa de Graduação em Ciências da Computação para a obtenção do Grau de Bacharel em Ciências da Computação.

Florianópolis - SC, Brazil, 8 de dezembro de 2020:

Profª. Dra. Jerusa Marchi
Orientadora

Prof. Dr. Eduardo Inácio Duzzioni
Banca

MSc. Otto Menegasso Pires
Banca

Florianópolis - SC, Brazil
2020

Dedico esse trabalho aos meus pais que me apoiaram do início ao fim e aos professores que contribuíram repassando seu conhecimento.

Agradecimentos

Para minha família e amigos.

Resumo

Computadores quânticos são cada vez uma realidade menos distante e, com eles, vem a grande expectativa de melhorar a eficiência de algoritmos que buscam resolver diversos tipos de problemas considerados intratáveis classicamente. Neste sentido, será feito neste trabalho um estudo sobre modelos teóricos de computação e as principais classes de complexidade clássicas e quânticas, mostrando relações de contingência e equivalência entre elas, de forma a definir os limites da computação quântica sobre aspectos teóricos de computabilidade e tratabilidade. Definindo também as principais relações em aberto e conjecturas na literatura sobre a teoria da complexidade computacional quântica.

Palavras-chave: Computação Quântica. Classes de Complexidade. Bounded error Quantum Polynomial Time.

Abstract

Quantum computers are increasingly less distant reality and, with them, comes a great expectation of improving the efficiency of algorithms that seek to solve different types of problems considered classically intractable. In this sense, a study on theoretical models of computation and the main classical and quantum complexity classes will be done, showing contingency and equivalence relations between them, in order to define the limits of quantum computing on theoretical aspects of computability and treatability. Also defining the main open relationships and conjectures in the literature on the theory of quantum computational complexity.

Keywords: Quantum Computing. Complexity classes. Bounded error Quantum Polynomial Time.

Sumário

1 Introdução

Até meados de 1965 não havia nenhuma previsão real sobre o futuro do hardware quando Gordon E. Moore fez sua profecia, na qual o número de transistores dos chips teria um aumento de 100%, pelo mesmo custo, a cada período de 18 meses. Essa profecia tornou-se realidade e acabou ganhando o nome de Lei de Moore (DISCO; MEULEN, 1998).

Segundo Nielsen e Chuang (2010), à medida que a escala dos transistores diminuía, ela se aproximava cada vez mais de limites físicos fundamentais. Nos dias atuais, empresas como Intel e Nvidia apostam em outras alternativas aos chips de silício, investindo cada vez mais na Computação Quântica e também em Computação Biológica, para superar este obstáculo.

A Computação Quântica é um novo paradigma de computação em que se utilizam sistemas quânticos para se realizar processamento de informação. E nela são introduzidos recursos não existentes na Computação Clássica. (POLLACHINI, 2018).

Segundo Melo e Christofoletti (2003), a computação quântica é uma proposta para realizar o processo da computação usando álgebra quântica. A mecânica quântica é fundamental nesse processo. O computador clássico opera com uma sequência de zeros e uns, de modo que qualquer ação computacional pode ser traduzida em última instância por uma sequência desses algarismos. E esses zeros e uns, são na verdade estados lógicos de transistores. Não é possível ter os dois ao mesmo tempo.

Já os computadores quânticos mantém um conjunto de qubits. Um qubit pode conter um 1, um 0 ou uma sobreposição destes. Em outras palavras, pode conter tanto um 1 como um 0 ao mesmo tempo. O computador quântico funciona pela manipulação destes qubits. Com isso, uma coleção de qubits poderia representar uma fileira de números ao mesmo tempo, e um computador quântico poderia processar toda uma entrada de dados simultaneamente (MELO; CHRISTOFOLETTI, 2003).

Com isso, surgem as questões sobre o poder computacional de um computador quântico, que motivam a produção deste trabalho. Seriam computadores quânticos capazes de solucionar problemas que não podem ser solucionados por um modelo clássico de computação, como uma máquina de Turing? Seriam os computadores quânticos capazes de tratar problemas que hoje são considerados intratáveis nos modelos de computação clássicos? Quais as classes de problemas que podem ser efetivamente resolvidos com a utilização de algoritmos quânticos? Quais são as relações entre as classes de complexidade clássicas como por exemplo, P e NP, com as classes de complexidade quânticas?

Segundo Pollachini (2018), a expectativa é que um modelo híbrido entre Computação Clássica e Quântica seja a tecnologia emergente no cenário atual; A computação seria realizada em um processador quântico para problemas nos quais há vantagens no uso da Computação Quântica, como espera-se que ocorra com os problemas chamados NP-difíceis. Dentre as principais aplicações previstas para a Computação Quântica, destaca-se resolução de problemas de otimização, *machine learning*, desenvolvimento de novos materiais, fármacos e processos químicos.

O algoritmo quântico de Shor (1997), possibilita que fatorações de números primos e o cálculo de logaritmos discretos sejam feitos de forma eficiente, impactando assim sistemas de criptografias que se baseiam nessa dificuldade prática, como por exemplo, o algoritmo de criptografia assimétrica RSA, publicado por Rivest, Shamir e Adleman (1978) e amplamente ainda utilizado até hoje.

Dadas as aplicações futuras e expectativas desse novo paradigma surge a necessidade de pesquisa em ciência de base, preocupada em responder como as coisas se relacionam e quais são as potencialidades da computação quântica, no caso, como se organizam as classes de complexidade quânticas e quais são suas relações com as outras classes clássicas.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Este trabalho de conclusão de curso tem por objetivo, realizar uma pesquisa científica de base, para apresentar a definição de classes de complexidade clássicas, probabilísticas e quânticas presentes na literatura, assim como apresentar e definir as relações entre as classes de complexidade que já são conhecidas atualmente e mostrar as que ainda estão em aberto, definindo um panorama geral do estado da arte da pesquisa nessa área, baseando-se nas ideias de algoritmos e computação/verificação de soluções.

1.1.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

- Apresentar os conceitos fundamentais que viabilizam o estabelecimento das classes de complexidades clássicas e probabilísticas, notadamente as definições de Máquinas de Turing determinísticas e probabilísticas;
- Estudar, compreender e apresentar as principais classes de complexidade clássicas e probabilísticas;
- Estudar e compreender a Computação Quântica de Circuitos e seus principais conceitos;

- Apresentar e definir a Máquina de Turing Quântica e seu funcionamento;
- Estudar e compreender as principais classes de complexidade quânticas;
- Apresentar e referenciar os principais resultados e provas referentes às relações entre as classes de complexidade clássicas, probabilísticas e quânticas.

1.2 Apresentação do trabalho

Este trabalho será apresentado da seguinte maneira: o Capítulo 2, sobre Computação Clássica, inicia apresentando conceitos básicos de Máquina de Turing, suas variantes não determinística e probabilística, em sequência define classes de complexidade clássicas, algumas amplamente conhecidas como: *P, NP, EXP e PSPACE*, classes probabilísticas como: *BPP e PP* e outras que envolvem protocolos de comunicação como: *IP, MA e AM*, indicando problemas relevantes que pertencem as mesmas.

Ainda no mesmo capítulo serão mostradas as relações de equivalência e contingência provadas na literatura envolvendo as classes clássicas apresentadas, e de forma iterativa será construído um diagrama que representa estas relações.

O capítulo 3, inicia apresentando conceitos básicos de computação quântica, necessários para compreender e definir os modelos de computação quântica de circuitos e a variante quântica da Máquina de Turing, no modelo de circuito serão apresentadas algumas portas lógicas e circuitos necessários para definir classes de complexidade quânticas, e na seção da variante quântica da Máquina de Turing, é apresentada a sua definição formal e seu funcionamento.

Em sequência no mesmo capítulo serão definidas as principais classes de complexidade quânticas BQP, QMA e $QIP(k)$, através dos modelos apresentados, indicando problemas que pertencem a essas classes e, por fim, demonstrando suas relações de equivalência e contingência com todas as outras classes apresentadas, clássicas e quânticas.

O Capítulo 4, apresenta algumas implicações e pontos em aberto nos relacionamentos entre classes. E ao final, no Capítulo 5, apresentam-se as considerações finais do trabalho.

2 Computação Clássica

Neste capítulo estão descritos os conceitos que formam a base do estudo das classes de complexidade clássicas, como os principais modelos de computação que são utilizados para definir classes de complexidade, definições formais de classes de complexidade e relações entre as mesmas. As relações podem ser de equivalência, contenção ou até mesmo serem desconhecidas, baseando-se em provas formais encontradas na literatura.

2.1 Máquina de Turing

Dentro dos diversos modelos de dispositivos de computação, a Máquina de Turing proposta em 1936 por Alan Turing, matemático britânico considerado o pai da Ciência da Computação, é um modelo teórico de um computador que possui uma fita única representando uma memória ilimitada. Segundo a tese de Church (1936) e Turing (1937), todo processo efetivo (para o qual existe um algoritmo, ou um processo mecânico de computação) pode ser efetuado por meio de uma Máquina de Turing.

Segundo Sipser (2006) até hoje esta tese não foi refutada e o modelo de computação é considerado o mais poderoso em termos de reconhecimento de linguagens, sendo equivalente às gramáticas tipo-0 na hierarquia de Chomsky e reconhecendo as linguagens recursivamente enumeráveis.

Uma máquina de Turing pode ser definida formalmente, segundo Sipser (2006, p. 148) como uma 7-upla $(Q,\ \Sigma,\ \Gamma,\ \delta,\ q_0,\ q_{aceita},\ q_{rejeita})$ onde $Q,\ \Sigma$ e Γ são todos conjuntos finitos e:

1. Q é o conjunto de estados.
2. Σ é o alfabeto de entrada não contendo o símbolo em branco.
3. Γ é o alfabeto de fita, onde $\sqcup \in \Gamma\ e\ \Sigma \subseteq \Gamma$.
4. $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times$ {E,D} é a função de transição.
5. $q_0 \in Q$ é o estado inicial.
6. $q_{aceita} \in Q$ é o estado de aceitação.
7. $q_{rejeita} \in Q$ é o estado de rejeição, onde $q_{rejeita} \neq q_{aceita}$.

Ainda segundo Sipser (2006), a máquina faz a computação de uma entrada $w \in \Sigma^*$, onde Σ^* representa o conjunto infinito de palavras obtidas a partir dos símbolos de Σ, começando com uma cabeça de leitura/escrita na posição mais à esquerda da fita e seguindo

a função de transição definida, podendo deslocar a cabeça à esquerda ou à direita em cada passo, a computação se encerra ao entrar em um estado de aceitação ou rejeição, caso não aconteça, a máquina continua para sempre.

2.1.1 Máquina de Turing Não-determinística

Segundo Hopcroft, Motwani e Ullman (2001), uma Máquina de Turing Não-determinística difere de uma determinística por ter a função de transição que para cada estado $q \in Q$ e símbolo na fita $\gamma \in \Gamma$, a transição $\delta(q, \gamma)$ leva a um conjunto de triplas dado por $\{(q_1, \gamma_1, D_1), (q_2, \gamma_2, D_2), ..., (q_k, \gamma_k, D_k)\}$ sendo k um inteiro finito e D_i uma direção de movimento do cabeçote. Neste modelo da Máquina de Turing, para cada transição, pode-se escolher qualquer uma das triplas para transitar.

Como descrito por Sipser (2006), essa variante aceita uma entrada w se existir pelo menos uma sequência de escolhas de transições que conduzam a máquina para um estado de aceitação. Apesar dessa variante parecer ser mais potente que uma Máquina de Turing Determinística, como uma Máquina de Turing Não-determinística pode ser simulada por uma Determinística, as duas aceitam exatamente o mesmo conjunto de linguagens, como mostrado por Sipser (2006, 122). Porém, tal simulação na Máquina de Turing Determinística pode levar um tempo exponencialmente maior que na variante Não-determinística.

2.1.2 Máquina de Turing Probabilística

Segundo Sipser (2006, 393), uma Máquina de Turing Probabilística M é um tipo de Máquina de Turing Não-determinística na qual cada passo não-determinístico é chamado passo de arremesso-de-moeda e tem dois próximos movimentos legítimos. Para cada um dos ramos de computação é atribuída uma probabilidade e a probabilidade de M aceitar uma entrada w será o somatório das probabilidades de cada ramo de aceitação.

A Máquina de Turing Probabilística pode ser definida formalmente, como por Du e Ko (2014), como sendo uma 7-upla $(Q,\ \Sigma,\ \Gamma,\ \delta,\ q_0,\ q_{aceita},\ q_{rejeita})$, tendo os mesmos componentes que uma Máquina de Turing Determinística, exceto pela função de transição δ que é definida da seguinte forma:

$$\delta : Q \times \Gamma \times Q \times \Gamma \times \{E, D\} \rightarrow [0, 1]$$

Nesta função de transição, a máquina define para cada estado $q \in Q$, e símbolo na fita $\gamma \in \Gamma$, para qual estado a máquina deve ir, escrevendo um símbolo na fita e deslocando a cabeça de leitura para a esquerda ou direita, mapeando cada conjunto possível a um valor que indica a probabilidade, entre 0 e 1, de ocorrer.

A computação nesta máquina é similar a uma máquina de Turing Determinística, porém em cada passo de computação podemos utilizar um gerador uniforme de números aleatórios entre 0 e 1 para que a máquina escolha uma próxima configuração válida, dentre todas as possíveis. A probabilidade da máquina aceitar uma cadeia w é a soma das probabilidades de todos os caminhos possíveis para a máquina ir do estado inicial para um estado de aceitação.

Podemos demonstrar a computação com um exemplo, como na Fig. 1, sendo os vértices C_k configurações de uma Máquina de Turing Probabilística, e as arestas entre vértices indicando que é possível ir de uma configuração à outra em um passo de computação, cada aresta é valorada com uma probabilidade da máquina efetuar a transição.

Se considerarmos C_1 uma configuração inicial da máquina para uma dada entrada, e C_5 a configuração de aceitação. A probabilidade de aceitarmos a entrada pelo caminho $C_1 \to C_2 \to C_5 = 0.07$, e pelo caminho $C_1 \to C_3 \to C_5 = 0.12$. Logo a probabilidade p de aceitarmos a cadeia é: $p = 0.07 + 0.12 = 0.19$.

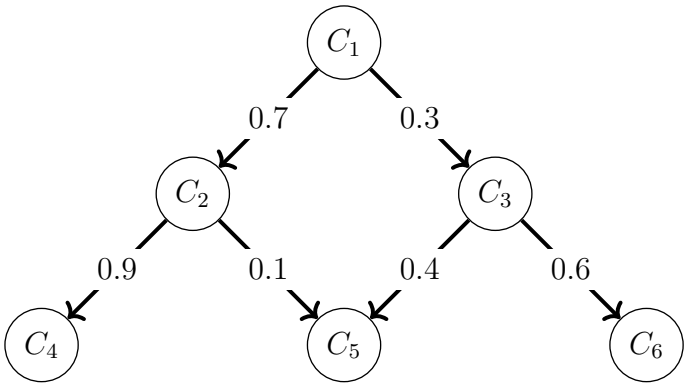

Figura 1 – Exemplo de computação em uma MTP.

Uma definição alternativa e equivalente é mostrada por Arora e Barak (2009), que afirma que, sintaticamente, uma Máquina de Turing Probabilística não é diferente de uma Máquina de Turing Não-determinística: Ela é uma Máquina de Turing com duas funções de transição δ_0, δ_1, e a diferença está em como interpretamos o grafo de todas as possíveis computações, em vez de perguntar se existe uma sequência de escolhas que faz a Máquina de Turing aceitar, perguntamos quão grande é a fração de escolhas na qual isso acontece. Mais precisamente, para cada passo na computação, a máquina decide aleatoriamente qual função de transição irá aplicar.

2.2 Complexidade de tempo

Dado um modelo computacional abstrato que resolve problemas computacionais, como uma Máquina de Turing e suas variantes, é possível ser feito um estudo sobre o

tempo necessário para se resolver um determinado problema neste modelo computacional. Visando classificar diferentes problemas sobre os aspectos de complexidade temporal foram estabelecidas diversas classes de complexidade que agrupam problemas diferentes que podem ser resolvidos em tempos semelhantes. Segundo Sipser (2006, p. 264), seja M uma Máquina de Turing Determinística que finaliza sobre todas as entradas. O tempo de execução ou complexidade de tempo de M é a função $f : N \to N$, onde $f(n)$ é o número máximo de passos que M usa sobre qualquer entrada de comprimento n. Partindo dessa definição é possível definir diversas classes de complexidade, as principais serão vistas nas subseções seguintes.

2.2.1 Classe P

A classe P, também conhecida como *Polynomial-Time* é uma das classes de complexidade fundamentais, definida inicialmente por Edmonds (1965), Cobham (1964) e Rabin (1960). A classe P contém todos os problemas de decisão que podem ser decididos por uma Máquina de Turing Determinística usando uma quantidade polinomial de tempo de computação, ou seja existe um polinômio $p(n)$, sendo n o comprimento da entrada, em que a máquina sempre para e decide após $p(n)$ passos.

Com as contribuições de Cobham (1964) e Edmonds (1965) podemos assumir que os problemas serão considerados tratáveis, ou seja, que podem ser viáveis e eficientes de serem decididos em algum dispositivo computacional, somente se podem ser computados em tempo polinomial por um Máquina de Turing Determinística.

Sipser (2006) também mostra que essa classe contém além de outros problemas, todos os problemas que podem ser decididos por Autômatos Finitos, e Autômatos de Pilha com somente uma pilha, partindo da demonstração que toda Linguagem Livre de Contexto, é um membro de P, e o conjunto de linguagens decididas por um Autômato de Pilha é o mesmo que o conjunto de Linguagens Livres de Contexto.

Um exemplo de problema com aplicação prática e que também pertence a essa classe é o de encontrar uma árvore geradora mínima em um grafo conexo com pesos. Esse problema pode ser resolvido em tempo polinomial por dois algoritmos, que são os algoritmos de Prim (1957) e Kruskal (1956).

A complexidade do algoritmo de Prim depende da estrutura de dados utilizada para representar o grafo, no pior caso possui complexidade de tempo igual a $\mathcal{O}(m^2)$. Já o algoritmo de Kruskal possui complexidade de tempo igual a $\mathcal{O}(m \log n)$, onde m representa o número de arestas e n o número de vértices.

2.2.2 Classe NP

A classe NP, também conhecida por *Non-Deterministic Polynomial Time*, pode ser definida formalmente, como feito por Sipser (2006), sendo a classe de linguagens que uma Máquina de Turing Não-determinística consegue decidir em tempo polinomial ao tamanho da entrada, uma Máquina de Turing Determinística equivalente poderia levar tempo exponencial para decidir a mesma entrada, dado que podemos simular o não-determinismo percorrendo todos os ramos de computação deterministicamente, como visto na prova fornecida por Sipser (2006, 121), e por isso consideramos NP uma classe de problemas para os quais não temos soluções eficientes e consideradas tratáveis.

Sipser (2006) também demonstra outra formalização da classe NP como sendo a classe de linguagens que pode ser verificada em tempo polinomial. Ou seja, dada uma cadeia de entrada é possível verificar se ela pertence à linguagem ou não em tempo polinomial. Por exemplo ao tentar adivinhar uma senha aleatória pelo algoritmo de força bruta, testando todas as combinações de caracteres levaríamos um tempo super polinomial em uma Máquina de Turing determinística, porém para testar se uma senha é a correta ou não, levaria tempo polinomial.

Sipser (2006) define formalmente um Verificador, para uma linguagem L, sendo um algoritmo V, onde:

$$L = \{w \mid V \; aceita \; < w, c > \; para \; alguma \; cadeia \; c\}$$

sendo c um certificado ou prova que pode ser usado para verificar que uma cadeia w pertence a linguagem L. Se uma linguagem L possui um verificador que roda em tempo polinomial então $L \in NP$.

Boa parte dos problemas podem ser solucionados de formas inteligentes em tempo polinomial evitando uma busca por força-bruta, porém muitos problemas ainda resistem e suas soluções ainda levam tempo super polinomial. A questão se volta para o seguinte ponto: será que esses problemas são realmente mais difíceis que os problemas em P e não podem ser resolvidos em tempo polinomial? Ou será que o algoritmo que leva tempo polinomial para esses problemas ainda não foi descoberto?

Sabemos que P está contido em NP, pois todo problema decidido em tempo polinomial também é verificável em tempo polinomial. Porém a questão se $P = NP$ ainda está em aberto até hoje.

A classe NP contém uma extensa lista de problemas, os quais podem ser verificados em tempo polinomial. Podemos dar como exemplo o problema do isomorfismo de grafos, em que as entradas são dois grafos G_1 e G_2 não-dirigidos e queremos decidir, respondendo sim caso G_1 e G_2 sejam isomorfos, e respondendo não caso não sejam.

Dois grafos $G_1 = (V_1, A_1)$ e $G_2 = (V_2, A_2)$ são isomorfos se existe uma função bijetora $f : V_1 \to V_2$ que mapeia o conjunto de vértices de G_1 ao conjunto de vértices em G_2, de tal forma que, dois vértices i e j são adjacentes em G_1 se e somente se $f(i)$ e $f(j)$ são adjacentes em G_2.

$$\forall v_i, v_j \in V_1, (v_i, v_j) \in A_1 \leftrightarrow (f(v_i), f(v_j)) \in A_2$$

Descobrir que dois grafos são isomorfos não é uma tarefa trivial, um dos algoritmos mais eficientes é apresentado por Babai e Luks (1983) que tem tempo de execução $2^{O(\sqrt{n.logn})}$. Porém se recebermos como certificado a função bijetora, codificada em uma palavra de tamanho w, podemos verificar o isomorfismo em tempo polinomial.

Por exemplo, podemos verificar que os grafos na imagem a seguir são isomorfos, pois existe a função $f : V_1 \to V_2 = \{f(1) = 1, f(2) = 3, f(3) = 5, f(4) = 2, f(5) = 4\}$

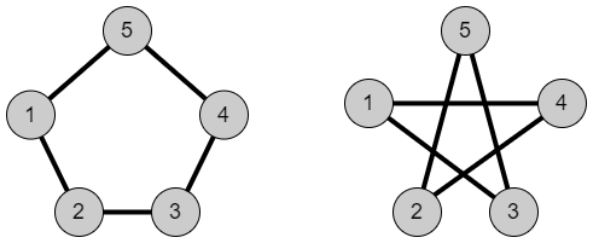

Figura 2 – Isomorfismo em grafos.

Diferentemente do problema do isomorfismo em grafos, até hoje o problema do não-isomorfismo em grafos não foi provado estar em NP, este problema busca responder sim se dados dois grafos $G_1 = (V_1, A_1)$ e $G_2 = (V_2, A_2)$ não são isomorfos. Note que, caso o certificado seja uma função bijetora que mapeia vértices de G_1 em G_2, verificar que existem adjacências em G_1 que não existem em G_2, não demonstra que os grafos não são isomorfos, pois pode ainda existir uma outra função bijetora que faça o mapeamento, mostrando que os grafos eram isomorfos.

2.2.2.1 NP-Completude

Dando uma luz na questão P *versus* NP, o trabalho de Cook (1971) e Levin (1973) mostrou que existem problemas em NP cuja complexidade é a mesma da classe inteira. Os chamados problemas $NP - Completos$ são considerados os problemas mais difíceis da classe; A implicação disso é que se existir, para qualquer problema $NP - Completo$, um algoritmo que o solucione em tempo polinomial, todos os outros problemas em NP também o serão.

Segundo Sipser (2006), os problemas $NP - Completos$ são importantes por muitas razões. Uma delas é que podemos provar que $P \neq NP$ mostrando que um problema em

NP requer mais tempo que polinomial, ou que $P = NP$ encontrando um algoritmo de tempo polinomial para um problema $NP - Completo$.

O conceito de completude de uma classe pode ser estendido para outras classes além de NP, veremos em outras seções sobre problemas $PSPACE - Completos$ ou $QMA - Completos$, que similarmente a classe dos problemas $NP - Completos$, também são provados serem os problemas mais difíceis de suas respectivas classes.

Boolean satisfiability problem, abreviado como SAT, foi o primeiro problema provado $NP - Completo$, por Cook (1971), e consiste em determinar se existem valores para variáveis, de uma fórmula booleana, que a façam ser verdadeira.

Podemos exemplificar o problema com uma fórmula w_1, que possui x_1, x_2, x_3 como variáveis booleanas e pode ser satisfeita por um certificado $c_1 = (a, b, c)$, onde a, b e c assumem valores 0 (falso) ou 1 (verdadeiro), que determine os valores das variáveis booleanas. c_1 pode ser verificado em tempo polinomial.

$w_1 = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge \neg x_1$

$c_1 = (0, 0, 0)$

Em tempo polinomial é possível mostrar que:

$w_1 = (0 \vee \neg 0) \wedge (\neg 0 \vee 0 \vee 0) \wedge \neg 0 = 1 \wedge 1 \wedge 1 = 1$

2.2.3 Classe BPP

Definida inicialmente por Gill (1977) a classe *Bounded-Error Probabilistic Polynomial Time* - BPP é a classe em que todos os problemas de decisão são solucionados em tempo polinomial com erro máximo de 1/3. Ou seja, uma cadeia de entrada que pertence a linguagem tem no mínimo probabilidade 2/3 de ser aceita pela máquina, e se a cadeia não pertence a linguagem então a máquina a rejeita com probabilidade pelo menos 2/3.

A escolha da probabilidade de erro 1/3 é arbitrária. Na verdade, para qualquer valor de erro x constante, sendo $0 \leq x < 1/2$, a classe BPP será a mesma. A ideia por trás disso é que devido ao limite de Chernoff (1952), executar um algoritmo com erro constante várias vezes faz com que a probabilidade da resposta errada ser maioria decaia exponencialmente.

Podemos definir BPP, alternativamente, como por Arora e Barak (2009), utilizando Máquinas de Turing determinísticas, onde as escolhas probabilísticas a serem aplicadas em cada etapa de computação podem ser fornecidas à máquina como uma entrada adicional e randômica r.

BPP contém uma linguagem L se existe uma Máquina de Turing M que executa em tempo polinomial, e um polinômio $p : \mathbb{N} \to \mathbb{N}$, $\forall w \in \{0, 1\}^*$, sendo w uma cadeia de

entrada, de tal modo que:

$Pr_{r \in R\{0,1\}^{p(|w|)}}[M(w,r) = L(w)] \geq 2/3$, ou seja a probabilidade da máquina M recebendo a entrada w e uma entrada randômica r, de tamanho polinomial à entrada w, responder corretamente se $w \in L$ ou $w \notin L$ deve ser maior que 2/3.

2.2.4 Classe PP

Definida por Gill (1977), a classe PP (*Probabilistic Polynomial-Time*) é a classe dos problemas de decisão que são decididos por uma Máquina de Turing em tempo polinomial ao tamanho da entrada com erro menor que 1/2 para todas as respostas. Ou seja, caso a resposta seja sim o algoritmo irá responder sim com probabilidade maior que 50%, caso seja não, a máquina responderá sim com probabilidade menor ou igual a 50%.

De modo intuitivo podemos mostrar que a classe $BPP \subseteq PP$, dado que os problemas da classe BPP tem erro menor que 1/3 para todas as instâncias, e a classe PP, como definida, contém todos os problemas com erro menor que 1/2.

Diferentemente da classe BPP, o erro na classe PP não precisa ser constante, por exemplo, podemos ter uma probabilidade de acerto de $1/2 + 1/2^n$ sendo n o tamanho da entrada, como foi descrito na subseção anterior, para problemas de BPP, é possível fazer amplificação de probabilidade com um pequeno número de repetições, enquanto que para problemas gerais de PP, não é possível. Já que para um valor grande de n a probabilidade ainda será muito próxima de 50%.

Essa distinção permite categorizar a classe BPP como tendo algoritmos eficientes, pois os executa em tempo polinomial, similar a classe P. Apesar de não haver nenhuma prova até o momento, é conjecturada ser igual a classe P.

Em contrapartida a classe PP é considerada ineficiente, já que problemas em NP podem levar tempo exponencial para serem decididos, como mostrado por Sipser (2006), e $NP \subseteq PP$. A prova dessa relação, consiste em mostrar que o problema SAT, provado $NP - Completo$ por Cook (1971) e Levin (1973), pertence a PP.

Uma das formas de mostrar que o problema $SAT \in PP$ é adaptando a prova descrita por Goldreich (1999) da seguinte forma:

Dada uma fórmula booleana F com n cláusulas, $F(x_1, x_2, ..., x_n)$, temos que decidir se ela pode ser satisfeita, ou seja, dizer se existem valores para as cláusulas que tornem a fórmula verdadeira. O algoritmo em PP escolhe uniformemente e aleatoriamente valores para x de x_1 até x_n e verifica se a fórmula F é verdadeira.

Se F é verdadeira retornamos SIM, caso F seja falso retornamos SIM com probabilidade $\frac{1}{2} - \frac{1}{2^{n+1}}$ e NÃO com probabilidade $\frac{1}{2} + \frac{1}{2^{n+1}}$. Note que temos 2^n possibilidades de valores para as cláusulas.

Se a fórmula for insatisfazível o algoritmo responde SIM com probabilidade $\frac{1}{2} - \frac{1}{2^{n+1}} < \frac{1}{2}$.

Se a fórmula for satisfazível pelo menos uma das 2^n combinações será verdadeira, então a probabilidade do algoritmo responder SIM será a probabilidade de responder SIM para caso F seja avaliada como falsa (no máximo $2^n - 1$ casos) multiplicado pela probabilidade de responder SIM para caso F seja avaliada como verdadeira (no mínimo 1 caso), logo:

$$Pr = \left(\frac{1}{2} - \frac{1}{2^{n+1}}\right) \cdot \left(1 - \frac{1}{2^n}\right) + 1 \cdot \frac{1}{2^n} = \frac{1}{2} + \frac{1}{2^{2n+1}} > \frac{1}{2}$$

Concluindo que $SAT \in PP$, desta forma, como todos os problemas em NP podem ser reduzidos ao problema SAT, então todos os problemas NP pertencem também a PP.

2.2.5 Classe IP e Sistemas de Prova Interativa

A noção de sistemas de prova interativa surgiu com Goldwasser e Sipser (1986) generalizando os protocolos Arthur-Merlin introduzidos por Babai (1985) e que serão também referenciados nas próxima Subseções. Segundo Sipser (2006), o desenvolvimento dos sistemas de prova interativa tem afetado profundamente a teoria da complexidade e tem levado a avanços importantes nos campos de criptografia.

Nos sistemas de prova interativa existem duas entidades: um Provador que possui recursos ilimitados de computação e que descobrirá se determinada sentença pertence a uma dada linguagem, e um Verificador que irá checar a resposta do Provador.

O Provador tentará convencer o Verificador do pertencimento de uma sentença em uma linguagem, porém o Verificador só possui tempo polinomial ao tamanho da entrada para tomar uma decisão.

A classe *Interactive Proof Systems* - IP, definida por Goldwasser, Micali e Rackoff (1985), é a classe dos problemas de decisão que são verificados por um protocolo interativo, como descrito acima. O Provador e o Verificador podem ter um número polinomial de rodadas de interação, por isso podemos chamar esta classe de $IP[poly]$.

No final do algoritmo do Verificador, se resposta for "sim", então o Provador deve ser capaz de fornecer um certificado para o Verificador, que o convença a aceitar com probabilidade $\geq \frac{2}{3}$. E se a resposta for não, então não importa o certificado que o Provador envie, o verificador sempre rejeitará com probabilidade $\geq \frac{2}{3}$.

Um clássico problema pertencente a classe NP que ilustra esse conceito é o problema do isomorfismo de grafos. Dois grafos G_0 e G_1 são isomorfos se os nodos de G_0 podem ser reordenados para que sejam idênticos aos do grafo G_1, como apresentado na seção 2.2.2.

O Verificador pode mandar os grafos G_0 e G_1 para o Provador, que possui recursos ilimitados para computar todas as permutações possíveis dos nodos e decidir se os grafos

são isomorfos. O provador, então, pode responder um mapeamento dos nodos de G_0 para os nodos de G_1, convencendo o Verificador que pode em tempo polinomial checar o resultado.

O problema do não-isomorfismo em grafos (que não sabemos se está em NP) também pode ser resolvido com um sistema de provas interativas, como especificado por Goldreich, Micali e Wigderson (1991). Sejam G_0 e G_1 dois grafos, o algoritmo funciona da seguinte forma:

1. O Verificador escolhe um bit b (0 ou 1) de forma aleatória, e uma permutação aleatória dos vértices do grafo G_b, enviando-a para o Provador.
2. O Provador responde com 0 se o grafo permutado é isomorfo ao grafo G_0, ou responde 1 se o grafo permutado é isomorfo ao grafo G_1.
3. O Verificador aceita se o Provador responder com bit original b

Pode-se notar que caso os grafos não sejam isomorfos, o provador sempre consegue responder corretamente, porém caso eles sejam realmente isomorfos, o provador não saberá qual dos grafos foi usado para gerar a permutação, dessa forma, somente pode fazer o Verificador aceitar com probabilidade 1/2.

2.2.6 Classe MA

A nome imaginativo Merlin-Arthur surgiu com Babai (1985), Merlin é um mago onisciente com recursos computacionais ilimitados que deseja provar algo para o rei Arthur, que somente tem tempo polinomial para checar. Merlin irá enviar um certificado para tentar convencer o rei Arthur a aceitar, porém Merlin é desonesto e pode tentar fazer o rei aceitar falsas afirmações.

Pode se comparar a classe MA como sendo a versão probabilística da classe NP, a qual verifica em tempo polinomial um certificado. Segundo Babai (1985), MA é um conjunto de linguagens L contidas em $\{0,1\}^*$ onde, para cada uma delas, existe uma Máquina de Turing M, probabilística que computa em tempo polinomial para todas as entradas w. Caso w pertença a linguagem então existe um certificado c que $M(w,c)$ aceita com probabilidade no mínimo 2/3. Caso w não pertença a linguagem então para qualquer certificado c, $M(w,c)$ aceita com probabilidade máxima 1/3. Se mudarmos a probabilidade de 2/3 para 1 e a probabilidade 1/3 para 0, temos a classe NP, mostrando que $NP \subseteq MA$.

Formalmente, como definida por Babai (1985), uma linguagem $L \in MA$ se existir uma Máquina de Turing probabilística e polinomialmente limitada M e polinômios p, q que:

- se $w \in L$ então: $\exists c \in \{0,1\}^{q(|w|)}\, Pr_{r \in \{0,1\}^{p(|w|)}}(M(w,r,c) = 1) \geq 2/3$,

- se $w \notin L$ então: $\forall c \in \{0,1\}^{q(|w|)}\, Pr_{r \in \{0,1\}^{p(|w|)}}(M(w,r,c) = 1) \leq 1/3$.

Sendo c o certificado fornecido por Merlin e de tamanho limitado por um polinômio, r uma cadeia aleatória, também limitada polinomialmente, que é sorteada por Arthur e é usada para tomar decisões probabilísticas. E finalmente, $Pr(M(w,r,c) = 1)$ é a probabilidade de que a Máquina de Turing com as entradas w, r e c responda sim. Note que a cadeia r é privada de Merlin, logo pela definição, caso $w \in L$, Merlin deve conseguir encontrar um certificado que independente da cadeia aleatória de Arthur r a probabilidade de aceitar seja $\geq \frac{2}{3}$.

A prova que a classe $MA \subseteq PP$ não é trivial e é resultado do trabalho de Vereschchagin (1992).

2.2.7 Classe AM

A classe AM (Arthur-Merlin) ou $AM[2]$, também introduzida por Babai (1985) é composta por problemas de decisão que podem ser verificados em tempo polinomial por um protocolo Arthur-Merlin com duas mensagens. Ou seja há apenas um conjunto de consulta e resposta.

Arthur, o Verificador, joga algumas moedas aleatórias e envia o resultado dos seus lançamentos para Merlin, que é um Provador e responderá a Arthur com uma suposta prova. Arthur então decide se aceita ou rejeita utilizando apenas a resposta de Merlin e o resultado dos seus lançamentos de moedas gerados anteriormente.

Formalmente uma linguagem L está em AM se existir uma Máquina de Turing probabilística e polinomialmente limitada M e polinômios p, q que para cada entrada w de tamanho $n = |w|$,

- se $w \in L$ então: $Pr_{r \in \{0,1\}^{p(n)}}(\exists c \in \{0,1\}^{q(n)} M(w,r,c) = 1) \geq 2/3$,
- se $w \notin L$ então: $Pr_{r \in \{0,1\}^{p(n)}}(\forall c \in \{0,1\}^{q(n)} M(w,r,c) = 0) \geq 2/3$.

Sendo c a suposta prova fornecida por Merlin e de tamanho limitado por um polinômio, e r é a sequência aleatória, também limitada polinomialmente que é gerada e usada por Arthur, como vimos similarmente na classe MA. Note que neste caso r é gerado por Arthur e enviado para Merlin, logo Merlin pode tomar decisões com base em mais informações em comparação a classe MA.

A classe de complexidade $AM[k]$ é a generalização da classe AM com a troca de k mensagens entre Merlin e Arthur, com a última mensagem sendo sempre de Merlin pra Arthur.

Na realidade é possível provar que o problema do não-isomorfismo em grafos pertence a classe AM, mesmo que os lançamentos de moedas sejam públicos, como demonstrado por Goldwasser e Sipser (1986).

2.2.8 Classe EXPTIME

Segundo Papadimitriou (1994) a classe $EXPTIME$, ou simplesmente EXP, pode ser definida formalmente como:

$$EXP = TIME\left(2^{n^k}\right)$$

Sendo EXP o conjunto de todos os problemas de decisão que podem ser resolvidos por uma Máquina de Turing Determinística em tempo exponencial ao tamanho n da entrada para qualquer $k \in \mathbb{N}$.

2.3 Complexidade de espaço

Como vimos na seção anterior, problemas podem ser classificados quanto a complexidade de tempo que levam para ser decididos. Porém, utilizando uma Máquina de Turing Determinística, também podemos classificar problemas em termos do espaço, ou memória, que eles necessitam. Esta seção apresenta brevemente algumas das principais classes de complexidade de espaço, que possuem relações com as classes de complexidade de tempo apresentadas.

Segundo Sipser (2006) podemos definir que a complexidade de espaço de uma Máquina de Turing M, que finaliza seu processamento sobre todas as entradas, é a função $f : \mathbb{N} \to \mathbb{N}$, onde $f(n)$ é o número máximo de células de fita que M visita sobre qualquer entrada de comprimento n. Se M é uma Máquina de Turing Não-determinística, onde todos os ramos de computação param sobre todas a entradas, a complexidade de espaço é definida como sendo o número máximo de células de fita que M visita sobre qualquer ramo e qualquer entrada de comprimento n.

Como usualmente a complexidade de espaço é estimada usando notação assintótica, de acordo com Sipser (2006), sendo $f : \mathbb{N} \to \mathbb{R}^+$ uma função podemos definir as classes de complexidade de espaço $SPACE(f(n))$ e $NSPACE(f(n))$ da seguinte forma:

$SPACE(f(n)) = \{L \mid L$ é uma linguagem decidida por uma Máquina de Turing Determinística de espaço $O(f(n))\}$

$NSPACE(f(n)) = \{L \mid L$ é uma linguagem decidida por uma Máquina de Turing Não-determinística de espaço $O(f(n))\}$

2.3.1 Teorema de Savitch

O teorema de Savitch (1970) é um dos primeiros resultados sobre complexidade de espaço, que demonstra como Máquinas de Turing Determinísticas, que utilizam como espaço $f(n)$ células, são capazes de simular Máquinas de Turing Não-determinísticas através do uso somente de $f^2(n)$ espaços em sua fita, sendo n o tamanho da entrada. Formalmente Savitch (1970) provou que:

Para qualquer função $f : \mathbb{N} \to \mathbb{R}^+$, onde $f(n) \geq n$,

$$NSPACE(f(n)) \subseteq SPACE(f^2(n))$$

2.3.2 PSPACE e NPSPACE

Podemos definir a classe $PSPACE$ de forma análoga a classe P definida na seção 2.2.1. Segundo Sipser (2006), $PSPACE$ é a classe dos problemas de decisão que utilizam espaço polinomial em uma Máquina de Turing Determinística, em outras palavras:

$$PSPACE = \bigcup_k SPACE(n^k)$$

.

Também podemos definir a classe $NPSPACE$ como sendo a versão não determinística de $PSPACE$. Porém, um corolário do Teorema de Savitch é afirmar que $PSPACE = NPSPACE$, visto que o quadrado de qualquer polinômio é também um polinômio.

2.4 Relações entre Classes

Esta seção tem o objetivo de reunir as principais relações entre as classes apresentadas, de forma a definir hierarquicamente suas relações e apontar questões em aberto que ainda guiam as pesquisas em complexidade computacional.

Podemos representar visualmente estas relações com um diagrama, em que classes de complexidade estão representadas em níveis, uma conexão entre duas classes de complexidade em níveis diferentes indica uma relação de inclusão da classe do nível inferior na classe de nível superior. Duas classes no mesmo nível sem conexões indicam que, até o momento, não há nenhuma relação conhecida sobre essas classes.

2.4.1 P x NP

Por definição podemos afirmar que $P \subseteq NP$, dado que, todo problema que pode ser resolvido em tempo polinomial deterministicamente, também é resolvido em tempo

polinomial por uma Máquina de Turing não-determinística que não precisa utilizar o não-determinismo.

O problema de saber se $P = NP$ ou $P \neq NP$ é considerado por muitos o principal problema em aberto da Ciência da Computação. Não podemos afirmar que é possível resolver em tempo polinomial todos os problemas que podem ser verificados em tempo polinomial.

Podemos representar graficamente a relação da seguinte forma:

Figura 3 – Relação entre P e NP.

2.4.2 BPP x P x NP

Como visto na seção 2.2.3, BPP contém todos os problemas que são decididos em tempo polinomial com erro menor que 1/3 por uma Máquina de Turing Probabilística. É trivial notar que todos os problemas em P estão contidos em BPP, já que o erro máximo de uma Máquina de Turing Determinística é 0. Ainda existem muitos problemas conhecidos em BPP que não sabemos se estão em P, porém o número desses problemas está diminuindo cada vez mais e é conjecturado que P seja igual a BPP, porém até o momento esse problema também está aberto.

Já o relacionamento entre NP e BPP é completamente desconhecido, não podemos afirmar que $BPP \subseteq NP$ ou $NP \subseteq BPP$.

Podemos atualizar o diagrama para incluir a classe BPP da seguinte forma:

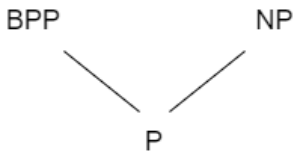

Figura 4 – Inclusão de BPP.

2.4.3 Relações com a classe MA

Definimos anteriormente, na seção 2.2.6, que na classe MA, Merlin envia um certificado para Arthur que verifica utilizando uma Máquina de Turing probabilística em tempo polinomial e responde com erro máximo de 1/3.

É imediato desta definição que $BPP \subseteq MA$, dado que Arthur pode ignorar a mensagem de Merlin e computar o problema ele mesmo, com sua Máquina de Turing probabilística e com erro máximo de 1/3.

Também é imediato que $NP \subseteq MA$, tendo em vista que, Arthur receberá o certificado de Merlin, e assim verificará em tempo polinomial e de forma determinística.

Não há qualquer resultado na literatura que prove que NP e BPP são subconjuntos próprios ou que as classes são equivalentes à classe MA, ou seja, estes problemas ainda permanecem em aberto.

Incluindo a classe MA no diagrama que ilustra as relações temos:

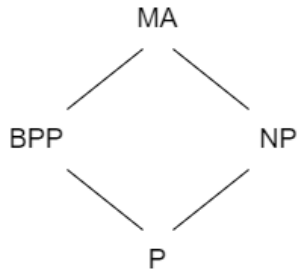

Figura 5 – Inclusão de MA.

2.4.4 Relações com a classe AM

Por definição, a classe MA está contida em $AM[2]$, tendo em vista que o protocolo de troca de mensagens entre Merlin e Arthur só necessita de uma mensagem de Merlin para Arthur em MA. Logo, todos os problemas em MA podem ser simulados pelo protocolo Merlin-Arthur com o uso de duas mensagens, em que a primeira mensagem, de Arthur para Merlin, é ignorada.

A classe $AM[k]$, definida similarmente a classe AM, porém com k rodadas de interação, foi provada por Babai e Moran (1988) ser equivalente a classe $AM[2]$ para todo $k > 2$.

Podemos incluir a classe AM no diagrama das relações da seguinte forma:

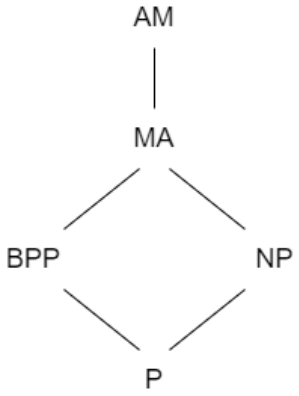

Figura 6 – Inclusão de AM.

2.4.5 Relações com a classe PP

É imediato que $BPP \subseteq PP$, pois algoritmos probabilísticos definidos em BPP, que possuem erro máximo constante $c < 1/2$, formam um subconjunto dentro de PP que define que o erro máximo não precisa ser constante porém sempre menor que $1/2$.

Também podemos mostrar que $NP \subseteq PP$, pois é possível provar que o problema da satisfação booleana, que é $NP - Completo$, pertence também a PP, uma das provas já foi apresentada na seção 2.2.4.

Porém, mais importante que a relação acima e incluindo a mesma, Vereschchagin (1992) mostrou que $MA \subseteq PP$. A ideia da prova é dado uma $L \in MA$ e q e M da definição formal de MA, podemos construir, usando uma amplificação padrão, com um novo polinômio p_1, uma Máquina de Turing Probabilística M_1 em que:

- se $w \in L$ então: $\exists c \in \{0,1\}^{q(n)}\, Pr_{r \in \{0,1\}^{p_1(n)}}(M_1(w,r,c) = 1) > 1 - 4^{-q(n)}$
- se $w \notin L$ então: $\forall c \in \{0,1\}^{q(n)}\, Pr_{r \in \{0,1\}^{p_1(n)}}(M_1(w,r,c) = 1) < 4^{-q(n)}$

Vereschchagin (1992) mostra que, considerando o par $\langle r, c\rangle \in \{0,1\}^{q(n)+p_1(n)}$, temos:

$$w \in L \Rightarrow Pr[M_1(w,r,c) = 1] > 2^{-q(n)}(1 - 4^{-q(n)}) > 4^{-q(n)}$$

$$w \notin L \Rightarrow Pr[M_1(w,r,c) = 1] < 4^{-q(n)}$$

Utilizando um resultado em que a constante $1/2$ da definição da classe PP pode ser substituída por qualquer outra constante ou número racional da forma $a(n)/2^{q(n)}$, sendo q um polinômio e $a : \{0,1\}^* \to \mathbb{N}$ uma função polinomialmente computável, com inteiros escritos em notação binária. Mostra-se que $L \in PP$.

Podemos adicionar PP ao diagrama da seguinte forma:

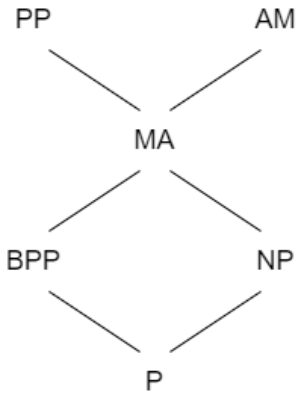

Figura 7 – Inclusão de PP.

2.4.6 Relações com a classe PSPACE

A relação $PP \subseteq PSPACE$ não é tão fácil de ser visualizada, uma maneira de entender essa relação, é notar que um algoritmo probabilístico que roda em tempo polinomial em uma Máquina de Turing Não-determinística, só necessita de, no máximo, espaço polinomial para cada ramo de computação.

Tendo em vista que podemos reutilizar o espaço após finalizar a computação de cada ramo, podemos passar por todas as configurações aleatórias possíveis, mantendo a contagem de quantos ramos aceitaram e rejeitaram. Para que a resposta seja sim, se mais de 1/2 dos caminhos de computação aceitaram, e seja não, no caso contrário, assim como a definição da classe PP.

Como resultado de Shamir (1990), a classe dos sistemas de provas interativas $IP[poly]$, com um número polinomial de rodadas de interação ao tamanho da entrada k, foi provada ser equivalente a classe $PSPACE$. E utilizando o resultado de Goldwasser e Sipser (1986) que $IP[k] \subseteq AM[k+2]$, mostramos que $IP[poly] = AM[poly] = PSPACE$, como $AM[k] \subseteq AM[poly]$ então $AM \subseteq PSPACE$.

Com as duas relações da classe $PSPACE$ apresentadas acima, podemos situá-la no diagrama de classes da seguinte forma:

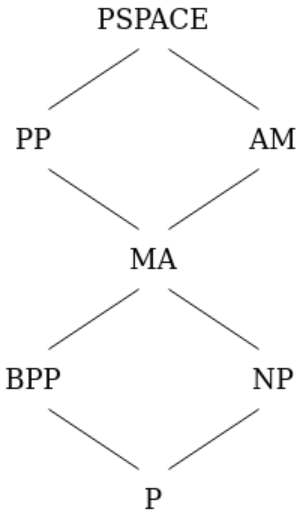

Figura 8 – Inclusão de PSPACE.

2.4.7 Relações com a classe EXPTIME

A relação $PSPACE \subseteq EXPTIME$ é relativamente simples de ser mostrada. Intuitivamente podemos pensar que um problema que é decidido por uma Máquina de Turing com espaço polinomial não pode levar tempo mais que exponencial para decidir o problema. Pois a Máquina de Turing terá um número exponencial de configurações, e dessa forma, teria que repetir alguma configuração em um determinado momento. Uma computação que decide não necessita entrar nesse ciclo, mostrando que com espaço polinomial não há como necessitar de tempo maior que exponencial.

Podemos provar formalmente, como mostrado em Sipser (2006, 308): Para $f(n) \geq n$, uma Máquina de Turing que usa $f(n)$ espaço pode ter no máximo $f(n)2^{O(f(n))}$ diferentes configurações. A computação por uma Máquina de Turing que decide não deve repetir uma configuração. Portanto uma Máquina de Turing que usa espaço $f(n)$ deve computar em tempo $f(n)2^{f(n)}$, então $PSPACE \subseteq EXPTIME$.

Podemos atualizar o diagrama de classes de complexidade para incluir $EXPTIME$:

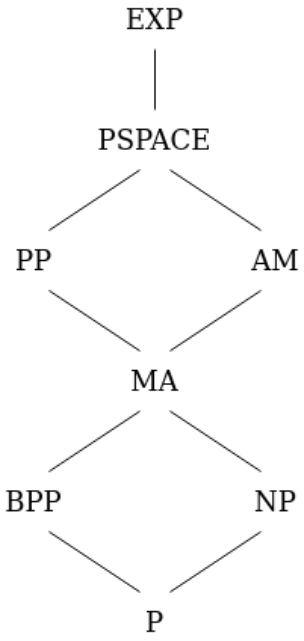

Figura 9 – Inclusão de EXPTIME.

3 Computação Quântica

Neste capítulo serão apresentados os principais conceitos para o entendimento de como são realizadas computações quânticas, tendo como referência, principalmente, o trabalho de Nielsen e Chuang (2010). Apresenta-se também os principais modelos de computação quântica e seu funcionamento, comparando-os com os modelos clássicos, assim como as principais classes de complexidade quânticas definidas sobre tais modelos de computação presentes na literatura, mostrando suas definições formais, problemas que pertencem a essas classes e relações com as classes definidas no capítulo anterior.

3.1 Conceitos fundamentais

3.1.1 Quantum bit

O conceito de *bit* na computação clássica é fundamental, sendo a menor unidade de medida usada para quantificar dados, o *bit* na computação clássica pode assumir apenas dois estados 1 e 0, ou verdadeiro e falso.

Segundo Nielsen e Chuang (2010), a computação quântica é construída sobre um conceito similar e análogo, o *quantum bit*, ou *qubit*. Assim como um *bit* clássico, o *qubit* pode assumir os estados $|0\rangle$ e $|1\rangle$, chamados de base computacional, que correspondem aos estados 0 e 1.

A *notação de Dirac* é a notação padrão da mecânica quântica para um vetor em um espaço vetorial, representado por $|\psi\rangle$, sendo ψ um rótulo comumente usado para um vetor, mas poderíamos usar qualquer outro. Matematicamente, veremos que um *qubit* é um vetor pertencente a um espaço vetorial, que tem como base $\{|0\rangle, |1\rangle\}$.

Diferente de um *bit*, um *qubit* pode formar uma combinação linear de estados, chamada de *superposição*, estando em um estado $|\psi\rangle$ diferente de $|0\rangle$ e $|1\rangle$:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$$

Sendo α e β números complexos e chamados de amplitude. Ao contrário de um *bit* clássico, não conseguimos examinar um *qubit* para determinar seu estado quântico, ou seja, os valores de α e β. Segundo a mecânica quântica, quando medimos um *qubit* o valor 0 é obtido com probabilidade $|\alpha|^2$, e o valor 1 é obtido com probabilidade $|\beta|^2$, naturalmente, como as probabilidades devem somar 1, então: $|\alpha|^2 + |\beta|^2 = 1$.

Vale ressaltar que um *bit* só pode ter estado 0 ou 1, enquanto que um *qubit* pode

existir em um estado contínuo entre $|0\rangle$ e $|1\rangle$ até ser medido ou observado. Por exemplo, um *qubit* pode estar no estado: $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$, com $|\frac{1}{\sqrt{2}}|^2 = 0.5$ de chance de dar o resultado 0 e 0.5 de chance de dar o resultado 1 ao ser medido. Após ser medido ele colapsa e todas as suas medições resultarão no mesmo resultado.

3.1.2 Esfera de Bloch

Podemos representar um *qubit* geometricamente através de um ponto na esfera de Bloch. Segundo Nielsen e Chuang (2010) ela provê uma maneira útil de visualizamos o estado de um único *qubit*, porém não há uma generalização da esfera de Bloch para múltiplos *qubits*.

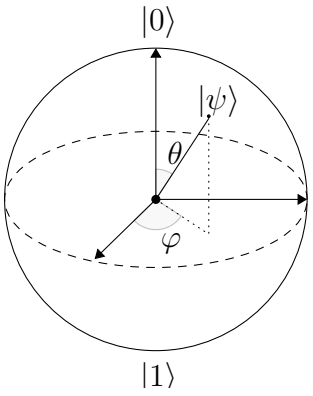

Figura 10 – Esfera de Bloch.

Neste modelo de representação, a base computacional $|0\rangle$ e $|1\rangle$ se encontra nos polos, os vetores $|\psi\rangle$ são representados por pontos na superfície da esfera e qualquer operação sobre um *qubit* representará na rotação do vetor.

3.1.3 Múltiplos qubits

Utilizando 2 *bits* clássicos podemos representar quatro estados: 00, 01, 10 e 11. Já utilizando 2 *qubits* teremos 4 estados em sua base computacional: $|00\rangle$, $|01\rangle$, $|10\rangle$ e $|11\rangle$, podemos ter uma superposição destes estados de tal forma que:

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle$$

Segundo Nielsen e Chuang (2010), de forma geral, considerando um sistema com n *qubits*, a base computacional de estados tem a forma $|x_1x_2...x_n\rangle$. Um estado quântico neste sistema é especificado por 2^n amplitudes. Da mesma forma, a probabilidade de obtermos um estado $|x\rangle$ é $|\alpha_x|^2$, e a soma de todas as probabilidades precisa ser 1.

3.2 Modelos de Computação Quântica

Nesta seção serão apresentados dois modelos de computação quântica que serão usados de base nas definições de classes de complexidade quânticas. O primeiro deles utiliza circuitos e portas lógicas quânticas e o segundo modelo descreve a versão quântica da Máquina de Turing Clássica. Segundo a prova dada por Yao (1993), qualquer função computável em tempo polinomial por uma máquina de Turing quântica tem um circuito quântico equivalente de tamanho polinomial. Sendo assim poderemos definir classes de complexidade quânticas na seção seguinte, a partir de qualquer um dos dois modelos.

3.2.1 Computação Quântica de Circuitos

Da mesma forma que podemos construir circuitos elétricos utilizando portas lógicas clássicas, como por exemplo com *NOT*, *OR* e *AND*, podemos construir circuitos quânticos com portas lógicas quânticas. Nessa subseção serão abordadas as principais portas lógicas quânticas e como a computação quântica circuital ocorre.

3.2.1.1 Portas Lógicas Quânticas

De modo análogo as portas lógicas clássicas, que transformam um conjunto de *bits* de entrada em um conjunto de *bits* de saída, podemos construir portas lógicas quânticas, que recebem *qubits* e realizam operações unitárias sobre os mesmos. Podemos representar uma porta lógica por um retângulo com um rótulo (Fig. 11), que deve conter o mesmo número de *qubits* de entrada e saída[1].

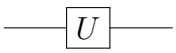

Figura 11 – Exemplo de porta lógica quântica.

Toda computação é realizada da esquerda para a direita e seu comportamento é descrito por uma matriz unitária. Podemos manipular um *qubit* usando sua representação em vetor coluna, dado que $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ então:

$$|\psi\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

.

Aplicamos uma operação sobre um *qubit* através da multiplicação do vetor coluna que o representa pela matriz da operação a ser realizada. Dentre as portas quânticas mais

[1] Diferentemente da computação clássica, a computação quântica é reversível pois não é possível dissipar energia no sistema, devido a isso, o número de entradas e saídas deve ser o mesmo.

notáveis temos a porta *NOT*, chamada de matriz de Pauli X, apresentada na Fig. 12 que funciona invertendo as amplitudes de probabilidade α e β.

$$\begin{bmatrix}\alpha\\ \beta\end{bmatrix} \times \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix} = \begin{bmatrix}\beta\\ \alpha\end{bmatrix}$$

Figura 12 – Operação da matriz Pauli X

Representada em circuitos como na Fig.13:

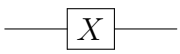

Figura 13 – Representação da porta quântica *NOT*.

Como visto anteriormente, operações sobre um *qubit* representam rotações do vetor sobre a superfície na esfera de Bloch. No caso da operação com a porta *NOT*, o vetor será rotacionado em 180° no eixo x. Além da Pauli X, outras duas portas, Pauli Y e Pauli Z, fazem parte das chamadas matrizes de Pauli. As mesmas realizam rotações de 180° do vetor no eixo y e z, respectivamente, na esfera de Bloch, e são representadas matricialmente como na Fig.14:

$$Y = \begin{bmatrix}0 & -i\\ i & 0\end{bmatrix} \qquad Z = \begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}$$

Figura 14 – Representação matricial das portas Pauli Y e Pauli Z.

E o resultado da aplicação destas portas sobre um vetor $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, ou seja, a multiplicação do vetor coluna pelas respectivas matrizes de operações, pode ser visto na Fig. 15:

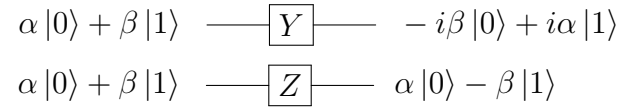

Figura 15 – Aplicação das portas Pauli Y e Pauli Z sobre um vetor.

A porta de Hadamard, por exemplo, é uma porta sem equivalência clássica, existindo apenas na computação quântica, que atua em um único *qubit*, podendo mapear o estado

inicial $|0\rangle$ para $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ e $|1\rangle$ para $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$ criando superposição. Representamos essa porta com o rótulo H, sua representação matricial é definida na Fig.16, e sua operação sobre um *qubit* $|\psi\rangle$, é definida como na Fig.17:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Figura 16 – Representação da porta de Hadamard.

$$\alpha\,|0\rangle + \beta\,|1\rangle \quad \text{——}\boxed{H}\text{——} \quad \alpha\frac{|0\rangle+|1\rangle}{\sqrt{2}} + \beta\frac{|0\rangle-|1\rangle}{\sqrt{2}}$$

Figura 17 – Aplicação da porta de Hadamard sobre um vetor.

3.2.1.2 Operações controladas

Além das portas de um único *qubit*, também podem ser construídas portas de múltiplos *qubits* que podem ser controladas por outros *qubits*. Uma porta controlada recebe dois *qubits*, um de controle e outro o alvo, e executa a operação sobre o alvo caso o controle possua valor $|1\rangle$. Por exemplo a porta *CNOT*, que recebe dois *qubits*, caso o *qubit* de controle seja $|1\rangle$ então ela fará a operação *NOT* no *qubit* alvo. Esta porta é representada pelo circuito na Fig.18, onde o *qubit* de controle é sinalizado com um ponto $\bullet$, o alvo pelo símbolo $\oplus$ e controle e o alvo são ligados por um fio.

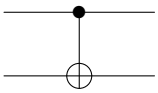

Figura 18 – Circuito da porta *NOT* controlada

Derivada da porta *CNOT*, é possível construir a porta de Toffoli (ou *CCNOT*) que, segundo Nielsen e Chuang (2010), é uma porta lógica universal reversível, ou seja, a partir dela é possível construir qualquer circuito reversível. O circuito é representado como na Fig.19 e pode ser construído fisicamente utilizando apenas portas lógicas de um *qubit* e portas *CNOT*.

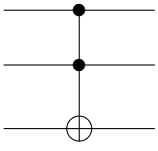

Figura 19 – Circuito da porta *Toffoli*

Esta porta recebe 3 *qubits* de entrada, sendo dois controles e um alvo, mais especificamente ela recebe (c_1, c_2, a) e mapeia para $(c_1, c_2, a \oplus (c_1.c_2))$. Com essa porta é possível criar uma porta lógica *NAND*, apenas colocando o valor alvo como 1 para obter como saída $(c_1, c_2, \neg(c_1.c_2))$.

Como a porta lógica *NAND* é uma porta lógica universal, ou seja é possível construir qualquer circuito clássico com um conjunto de apenas portas *NAND*, então Nielsen e Chuang (2010) apontam que os circuitos quânticos incluem os circuitos clássicos.

3.2.1.3 Famílias uniformes de circuitos quânticos

As famílias uniformes de circuitos quânticos podem ser uma forma alternativa de definirmos classes de complexidade quânticas através da computação quântica de circuitos.

Usualmente, classes de complexidade clássicas são definidas usando como base a definição de Máquina de Turing e a quantidade de tempo, ou espaço, necessárias para decidir ou verificar determinada linguagem. Porém também podemos definir classes de complexidade sobre famílias de circuitos. Como circuitos booleanos só permitem um número fixo de bits por entrada, podemos decidir uma linguagem através de um conjunto de circuitos $C_1, C_2, ...$ em que cada C_n recebe entradas de tamanho n.

Desta forma, uma linguagem L é decidida por uma família de circuitos $(C_i)_{i\in\mathbb{N}}$ se e somente se para toda entrada $w \in L$ de tamanho n, um circuito C_n aceita w.

Segundo Ruzzo (1979), uma família de circuitos booleanos é uniforme, se ela pode ser construída eficientemente por uma Máquina de Turing. Mais especificamente se existe uma Máquina de Turing probabilística rodando em tempo polinomial que gera a descrição do circuito C_n a partir de uma entrada 1^n.

Este conceito pode ser estendido para a computação quântica. Segundo Nishimura e Ozawa (1999), uma família quântica de circuitos é uma sequência infinita $C = \{C_n\}_{n\in\mathbb{N}}$ tal que C_n é um circuito quântico com entrada de tamanho n. E uma família quântica de circuitos é uniforme se uma Máquina de Turing Determinística clássica a descreve eficientemente.

3.2.2 Máquina de Turing Quântica

Uma Máquina de Turing Quântica, definida por Deutsch (1985), diferente das outras variantes de Máquinas de Turing pode encontrar-se numa superposição de configurações a cada instante, sendo a superposição de configurações uma combinação linear de configurações. Em uma Máquina de Turing Quântica é possível, a cada passo de computação, efetuar múltiplas transições nas diversas configurações. Potencialmente essa característica pode ser explorada para se obter algoritmos quânticos eficientes para problemas hoje considerados difíceis.

Ela também difere no sentido em que, ao final de sua execução, a máquina realiza uma medição, que consiste em escolher aleatoriamente uma das configurações e devido a essa característica probabilística, essa máquina pode produzir diferentes saídas para uma mesma configuração de entrada.

Segundo Ozawa (1998), uma Máquina de Turing Quântica $\mathcal{Q}$, é um sistema quântico consistido de um processador, uma fita infinita bilateralmente e um cabeçote para ler e escrever símbolos na fita. Sua configuração é determinada pela configuração q do processador, de um conjunto finito Q de estados, pela configuração da fita T representada por uma cadeia infinita de um conjunto finito Σ de símbolos, e a posição discreta do cabeçote ξ, sendo um valor no conjunto $\mathbb{Z}$ de inteiros.

Desta forma, qualquer configuração C de $\mathcal{Q}$ pode ser representada por uma tripla $C = (q, T, \xi)$ na configuração de espaço $Q \times \Sigma^* \times \mathbb{Z}$, sendo q o estado atual, T a configuração atual da fita e ξ a posição discreta do cabeçote sobre a fita. Assim o estado de $\mathcal{Q}$ é representado por um vetor unitário no espaço de Hilbert[2] $\mathcal{H}$ gerado pela configuração de espaço. A base computacional é representada por $|C\rangle = |q\rangle\,|T\rangle\,|\xi\rangle$ para qualquer configuração $C = (q, T, \xi) \in Q \times \Sigma^* \times \mathbb{Z}$ podendo escrever também como: $|q, T, \xi\rangle = |q\rangle\,|T\rangle\,|\xi\rangle$.

Assumimos que Q contém q_0 e q_f, que representam a configuração inicial e final do processador e Σ contém o símbolo B representando uma célula em branco na fita.

Para cada inteiro ξ o símbolo na fita na posição ξ é denotado por $T(\xi)$, assumimos que as configurações de fita possíveis são tais que $T(\xi) = B$ exceto para um número finito de células. O conjunto de possíveis configurações de fita é denotado $\Sigma^{\#}$ e é um conjunto contável.

A função de transição δ, pode ser definida, como por Bernstein e Vazirani (1997), com $\tilde{C}$ sendo um conjunto de amplitudes $\alpha \in \mathbb{C}$, no conjunto dos números complexos, no qual:

$$\delta : Q \times \Sigma \to \tilde{C}^{Q \times \Sigma \times \{E,D\}}$$

Dessa forma, dado um estado e um símbolo na fita, a função faz um mapeamento para um conjunto $\tilde{C}$. Cada elemento neste conjunto identifica uma amplitude e atrelada a cada amplitude está o próximo estado da máquina em Q, o símbolo em Σ que deve ser escrito na fita e define a direção do movimento do cabeçote podendo mover-se à esquerda (E) ou à direita (D). A função restringe que a soma dos quadrados das amplitudes seja igual a 1.

[2] Espaço de Hilbert é qualquer espaço vetorial que possua uma operação denominada produto interno, neste caso $\langle .|. \rangle$, e cuja métrica gerada por esse produto interno o torne um espaço completo em relação a norma, $||.|| = \sqrt{\langle .|. \rangle}$

A Máquina de Turing Quântica define um operador linear, chamado de operador de evolução temporal, se a Máquina $\mathcal{Q}$ começa numa configuração C com estado corrente q e símbolo a na fita, então após um passo $\mathcal{Q}$ vai estar em uma superposição de configurações, no estado $\psi = \sum_i \alpha_i C_i$, onde cada valor de amplitude $\alpha_i \neq 0$ foi definido na função de transição $\delta(q, a)$, sendo C_i a configuração da máquina após o processador ir para o estado definido na função de transição, escrevendo na fita onde o cabeçote se encontra e movendo o cabeçote em uma direção especificada. Estendendo esse mapeamento para o espaço inteiro temos o operador linear de evolução temporal U.

A computação consiste em preparar primeiramente o estado inicial $|C_0\rangle$ de tal forma que $|C_0\rangle = |q_0\rangle\,|T_{in}\rangle\,|0\rangle$, neste caso T_{in} é a entrada da máquina codificada e transformada em uma cadeia de entrada na fita, então a computação procede, chamada também de evolução do sistema, transformando o estado inicial no estado final. A dinâmica de $\mathcal{Q}$ é descrita por um operador unitário U em $\mathcal{H}$, que especifica a evolução de qualquer estado $|\psi(t)\rangle$ durante uma única etapa computacional no tempo t.

Um operador $U : \mathcal{H} \to \mathcal{H}$ é unitário se sua inversa coincidir com seu adjunto, ou de forma equivalente:

$$U^\top U = UU^\top = I$$

O operador de evolução age no estado $|\psi\rangle$ em t_0 e nos dá o estado $|\psi\rangle$ no tempo t:

$$|\psi(t)\rangle = U^t\,|\psi(0)\rangle$$

Ainda segundo Ozawa (1998), o resultado de uma computação é obtido fazendo uma medição na fita após a computação ter finalizado, diferente do caso clássico, a configuração da máquina não pode ser monitorada durante a computação por causa do efeito causado por uma medição.

Para isso Deutsch (1985) introduziu um único *qubit* adicional, chamado *qubit* de parada que é inicializado com $|0\rangle$ e cada algoritmo quântico define este *qubit* para $|1\rangle$ quando a computação finaliza, sem interagir com este *qubit* de outra forma. É possível observar este *qubit* para verificar se a computação finalizou.

3.2.3 Exemplo de Computação na Máquina

Nesta seção iremos exemplificar o funcionamento de uma Máquina de Turing Quântica que reconhece a linguagem regular 0^+1^+, como a expressão regular indica, uma cadeia pertence a essa linguagem se começar com uma sequência de um ou mais 0's, e em seguida, uma sequência de um ou mais 1's. Essa linguagem pode ser reconhecida, sem

dificuldades, por um autômato finito, mas por questões de simplicidade será utilizada neste contexto.

Para demonstrar o funcionamento utilizaremos uma notação simplificada, que tem a forma similar à $|\#000q_i11\#\rangle$, neste caso ela representa que o conteúdo da fita é $|\#00011\#\rangle$, o estado atual da máquina é $|q_i\rangle$, e o cabeçote se localiza na posição subsequente à posição do estado, ou seja, neste caso o cabeçote está sobre o primeiro valor 1 na fita. Adicionaremos o símbolo # para indicar o começo e o fim da cadeia de entrada. Também usaremos o símbolo $\vdash$ para indicar um passo de computação na máquina, no lado esquerdo, a configuração antes do passo, e no lado direito, a configuração após.

Podemos definir uma Máquina de Turing Quântica que reconhece 0^+1^+, da seguinte forma:

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_a, q_r\}$, sendo q_0 o estado inicial, q_a o estado de aceitação e q_r o estado de rejeição;

$\Sigma = \{0, 1\}$;

E com a função de transição definida na Tabela 1. Esta função leva em consideração que a entrada será pré-processada, adicionando o símbolo # no começo e fim da entrada, e o cabeçote da Máquina será posicionado no meio da cadeia de entrada[3].

δ	0	1	#
$\rightarrow q_0$	$\frac{1}{\sqrt{2}}^{(q_1,0,E)}, \frac{1}{\sqrt{2}}^{(q_2,0,D)}$	$\frac{1}{\sqrt{2}}^{(q_4,1,E)}, \frac{1}{\sqrt{2}}^{(q_5,1,D)}$	$1^{(q_r,0,E)}$
q_1	$1^{(q_1,0,E)}$	$1^{(q_r,1,E)}$	$1^{(q_a,0,D)}$
q_2	$1^{(q_2,0,D)}$	$1^{(q_3,1,D)}$	$1^{(q_r,0,E)}$
q_3	$1^{(q_r,0,D)}$	$1^{(q_3,1,D)}$	$1^{(q_a,0,E)}$
q_4	$1^{(q_6,0,E)}$	$1^{(q_4,1,E)}$	$1^{(q_r,0,D)}$
q_5	$1^{(q_r,0,D)}$	$1^{(q_5,1,D)}$	$1^{(q_a,0,E)}$
q_6	$1^{(q_6,0,E)}$	$1^{(q_r,1,E)}$	$1^{(q_a,0,D)}$

Tabela 1 – Função de transição da Máquina de Turing Quântica

Como já vimos anteriormente a função de transição mapeia um estado e um valor na fita para um conjunto de números complexos que indicam a amplitude em que a máquina irá para um determinado estado, escrevendo um valor na fita e movendo em uma direção seu cabeçote.

A ideia principal desta tabela de transições é começar no meio da cadeia e percorrer a fita nos dois sentidos ao mesmo tempo. Caso 0 seja o primeiro valor lido na cadeia então o ramo da esquerda só deve encontrar 0's até terminar, já o ramo da direita também pode encontrar mais 0, mas a partir de algum momento deve encontrar pelo menos um valor 1

[3] Como a fita é infinita bilateralmente, podemos, de forma equivalente, configurar a fita para que o centro da cadeia esteja na posição 0 na configuração inicial.

até chegar ao fim da cadeia de entrada, o mesmo raciocínio se aplica caso o primeiro valor lido seja 1.

Para exemplificar faremos uma simulação com duas entradas, a primeira $|00011\rangle$ que pertence à linguagem e a segunda $|00110\rangle$ que não pertence. Pré-processamos as entradas, como descrito anteriormente, e a configuração inicial da Máquina nos cenários será: $|\#00q_0011\#\rangle$ e $|\#00q_0110\#\rangle$.

Para a primeira entrada, a função de transição indica que $\delta(q_0, 0) = \frac{1}{\sqrt{2}}^{(q_1,0,E)}$, $\frac{1}{\sqrt{2}}^{(q_2,0,D)}$. Então, após um passo de computação teremos:

$$|\#00q_0011\#\rangle \vdash \frac{1}{\sqrt{2}}|\#0q_10011\#\rangle + \frac{1}{\sqrt{2}}|\#000q_211\#\rangle$$

Podemos ver que a máquina está em uma superposição de estados e em um próximo passo de computação, avaliaremos a função de transição tanto para $\delta(q_1, 0)$ quanto para $\delta(q_2, 1)$, o que resulta em:

$$\frac{1}{\sqrt{2}}|\#0q_10011\#\rangle + \frac{1}{\sqrt{2}}|\#000q_211\#\rangle \vdash \frac{1}{\sqrt{2}} \times 1\,|\#q_100011\#\rangle + \frac{1}{\sqrt{2}} \times 1\,|\#0001q_31\#\rangle$$

No próximo passo de computação, avaliaremos para $\delta(q_1, 0)$ e $\delta(q_3, 1)$, resultando em:

$$\frac{1}{\sqrt{2}}|\#q_100011\#\rangle + \frac{1}{\sqrt{2}}|\#0001q_31\#\rangle \vdash \frac{1}{\sqrt{2}} \times 1\,|q_1\#00011\#\rangle + \frac{1}{\sqrt{2}} \times 1\,|\#00011q_3\#\rangle$$

Por fim, será avaliada a função de transição em $\delta(q_1, \#)$ e $\delta(q_3, \#)$, que resulta em:

$$\frac{1}{\sqrt{2}}|q_1\#00011\#\rangle + \frac{1}{\sqrt{2}}|\#00011q_3\#\rangle \vdash \frac{1}{\sqrt{2}} \times 1\,|0q_a00011\#\rangle + \frac{1}{\sqrt{2}} \times 1\,|\#0001q_a10\rangle$$

Todos os ramos chegaram em um estado final, finalizando o processamento na máquina. Podemos efetuar uma medição no estado atual da máquina para saber se ela finalizou em um estado de aceitação ou rejeição. No caso desta sentença há $\frac{1}{2}$ de chance de obtermos $|q_a\rangle\,|000011\#\rangle$ e $\frac{1}{2}$ de obtermos $|q_a\rangle\,|\#000110\rangle$. Como, neste caso, o valor da fita é irrelevante para a tomada de decisão, aceitamos a entrada com probabilidade 1.

Para a segunda entrada, começaremos na configuração inicial $|\#00q_0110\#\rangle$ e seguiremos a função de transição. O processamento desta entrada será da seguinte forma:

$$|\#00q_0110\#\rangle \vdash$$

$$\frac{1}{\sqrt{2}}|\#0q_40110\#\rangle + \frac{1}{\sqrt{2}}|\#001q_510\#\rangle \vdash$$
$$\frac{1}{\sqrt{2}}|\#q_600110\#\rangle + \frac{1}{\sqrt{2}}|\#0011q_50\#\rangle \vdash$$
$$\frac{1}{\sqrt{2}}|q_6\#00110\#\rangle + \frac{1}{\sqrt{2}}|\#00110q_r\#\rangle \vdash$$
$$\frac{1}{\sqrt{2}}|0q_a00110\#\rangle + \frac{1}{\sqrt{2}}|\#00110q_r\#\rangle$$

Notamos que após a máquina finalizar seu processamento ela se encontra no estado $\frac{1}{\sqrt{2}}|q_a\rangle + \frac{1}{\sqrt{2}}|q_r\rangle$, logo como a entrada não pertence a linguagem então uma medição sobre o estado teria apenas 50% de chance de ser aceita erroneamente, isso porque somente o ramo da direita encontrou um erro, caso os dois ramos encontrassem erros então a probabilidade da entrada ser aceita cairia para 0%, por exemplo, se a entrada fosse $|1010\rangle$ e a configuração inicial $|\#10q_010\#\rangle$.

3.3 Complexidade Computacional Quântica

Nesta seção serão abordadas as principais classes de complexidade quânticas, definidas formalmente sobre os conceitos de circuitos quânticos e sobre o modelo abstrato de máquina de Turing quântica, indicando problemas pertencentes a estas classes.

3.3.1 Classe BQP

Segundo Bernstein e Vazirani (1997), *Bounded-error Quantum Polynomial-time* é a classe dos problemas (linguagens) de decisão que podem ser resolvidos em tempo polinomial por uma Máquina de Turing Quântica com erro probabilístico limitado a 1/3, para todas as cadeias de entrada. Geralmente é identificada como sendo a classe dos problemas tratáveis para computadores quânticos.

Uma outra definição formal equivalente a anterior à classe BQP é dada por Yao (1993). Sendo a classe BQP o conjunto de linguagens $L \subset \{0,1\}^*$ para a qual existe uma família uniforme de circuitos quânticos de tamanho polinomial $\{C_n\}$, ou seja, existe um algoritmo classicamente eficiente para produzir $\{C_n : n \in \mathbb{N}\}$ sobre uma base de portas universais que para todo n e entradas $x \in \{0,1\}^n$, onde $C_{|x|}$ é o circuito de tamanho $|x|$:

- Se $x \in L$, então $Pr[C_{|x|}(x) = 1] \geq 2/3$;
- Se $x \notin L$, então $Pr[C_{|x|}(x) = 1] \leq 1/3$.

Dessa forma, segundo Bernstein e Vazirani (1997) para uma entrada x existe um algoritmo probabilístico que gera a descrição de um circuito quântico, com a entrada *hard*

coded que decide se $x \in L$, ao final da computação o primeiro *qubit* será medido para efetuar a decisão sobre x.

Como visto em outras seções a escolha do valor do erro pode ser diferente de 1/3, e não altera a classe, desde que seja uma constante $k < 1/2$, pois devido ao limite de Chernoff (1952) podemos reduzir o erro exponencialmente rodando um número polinomial de vezes o algoritmo e aceitando o resultado majoritário.

O algoritmo proposto por Shor (1997) mostrou que o problema da fatoração de números inteiros que busca, dado um número inteiro, descobrir quais são os seus fatores primos e o problema do logaritmo discreto podem ser decididos em tempo polinomial e assim pertencem a esta classe de complexidade.

3.3.2 Classe QMA

Análogo a forma que a classe MA se relaciona com a classe BPP e a classe NP se relaciona com a classe P, a classe QMA se relaciona com a classe BQP. Essa classe, definida por Watrous (2000), contém todos os problemas de decisão que são verificados por uma Máquina de Turing Quântica em tempo polinomial com probabilidade de erro menor que 1/3, ou seja, caso a resposta da verificação do certificado for sim, então a probabilidade é mais de 2/3 do resultado ser verdadeiro. Caso o certificado não seja aceito então há no máximo uma probabilidade de 1/3 que a resposta deveria ser sim. Basicamente a classe QMA expande a definição de NP para permitir que a verificação seja realizada por uma computação quântica e o certificado possa ser um estado quântico.

QMA pode também ser entendida como: o conjunto de linguagens $L \subset \{0,1\}^*$ onde para cada linguagem existe um circuito verificador quântico V em tempo polinomial e para cada $x \in \{0,1\}^*$ com tamanho $n = |x|$, se $x \in L$ então existe um estado testemunha $|\xi\rangle$ de tamanho polinomial de *qubits* que $V(|x\rangle, |\xi\rangle)$ aceita com probabilidade maior ou igual a 2/3. Se $x \notin L$ então para todo estado testemunha $|\xi\rangle$, V aceita com probabilidade menor ou igual a 1/3 (BOOKATZ, 2014).

Formalmente, segundo Aharonov e Naveh (2002), uma linguagem $L \in QMA$ se existe um verificador quântico V executando em tempo polinomial e um polinômio p de tal forma que:

- Se $x \in L$ então, $\exists\, |\xi\rangle \in \mathcal{H}^{p(|x|)},\ Pr[V(|x\rangle\, |\xi\rangle) = 1] \geq 2/3$;
- Se $x \notin L$ então, $\forall\, |\xi\rangle \in \mathcal{H}^{p(|x|)},\ Pr[V(|x\rangle\, |\xi\rangle) = 1] \leq 1/3$.

Caso a sentença pertença a linguagem, então deve existir um estado testemunha

quântico[4] $|\xi\rangle$ de tamanho polinomial, tal que, o verificador aceite com probabilidade alta, recebendo o vetor de entrada e o estado testemunha. Caso não pertença então não deve existir tal certificado.

Diferentemente da classe NP, não há centenas de problemas conhecidos que pertencem a essa classe. Segundo Bookatz (2014), os problemas mais conhecidos podem ser divididos em três principais grupos: *Quantum circuit/channel property verification, Hamiltonian ground state estimation e Density matrix consistency.* Muitos desses problemas, se não todos, foram provados $QMA - completos$, ou seja, são os problemas mais difíceis da classe QMA.

3.3.3 Classe QIP

A classe QIP (Quantum Interactive Polynomial time), introduzida e definida formalmente por Watrous (1999), é a versão quântica da classe clássica IP. Pode ser informalmente descrita como sendo o conjunto de problemas de decisão, que podem ser verificados por um protocolo quântico de provas interativas. Nesta classe o verificador deve verificar através de algoritmos quânticos e em tempo polinomial, enquanto o provador tem recursos computacionais ilimitados. Neste protocolo podem ser trocadas um número polinomial de mensagens que podem ser estados quânticos.

Dado uma linguagem L e uma entrada x, é necessário que:

- se $x \in L$ então, o provador pode encontrar uma prova (ou certificado) que o verificador aceite com probabilidade no mínimo 2/3;
- se $x \notin L$ então, independente do certificado fornecido pelo provador, o verificador sempre rejeitará com probabilidade no mínimo 2/3.

Denominamos $QIP[k]$, para $k \in \mathbb{N}$, quando a troca de mensagens é limitada em uma constante k de mensagens, com o provador enviando a ultima mensagem.

Em uma versão preliminar no artigo de Watrous (1999), e mais tarde publicado também em Watrous (2003) provou também que $PSPACE \subseteq QIP[3]$, mostrando que o problema TQBF, um dos mais conhecidos problemas $PSPACE - completos$, pode ser resolvido por 3 rodadas de interação entre provador e verificador no protocolo descrito acima.

O problema TQBF (True Quantified Boolean Formula), é uma generalização do problema da satisfação booleana (SAT) em que cada variável é quantificada com operadores existenciais ou universais ($\exists$ ou $\forall$), e então é perguntado se dada uma fórmula sobre esse

[4] A classe de complexidade em que o verificador só tem acesso a provas clássicas é chamada de *Quantum Classical MA(QCMA)*, trivialmente $QCMA \subseteq QMA$.

conjunto de variáveis é verdadeira ou falsa. Mostrar que este problema está em $QIP[3]$ é suficiente para provar que existe um sistema quântico de provas interativas de 3 mensagens para todos os outros problemas de $PSPACE$, tendo em vista que estes problemas podem ser reduzidos em tempo polinomial ao problema TQBF.

Logo após a prova anterior, Kitaev e Watrous (2000) provaram que qualquer sistema quântico de provas interativas com um número polinomial de rodadas e com erro limitado bilateral pode ser paralelizado a um sistema de prova interativa quântica com erro unilateral exponencialmente pequeno no qual o provador e o verificador trocam apenas 3 mensagens. Dessa forma provando que: $\forall k > 3, QIP[k] = QIP[3] = QIP$.

3.3.3.1 QIP[1]

A classe $QIP[1]$ é equivalente a classe QMA por definição, dado que somente uma mensagem pode ser enviada, do provador para o verificador e será verificada por uma Máquina de Turing Quântica em tempo polinomial com erro máximo de 1/3.

3.3.3.2 QIP[2]

A classe $QIP[2]$, como já definida anteriormente, poderá trocar no máximo duas mensagens entre o provador e o verificador. Esta classe está trivialmente contida em $QIP[3]$ e contém trivialmente a classe $QIP[1]$.

3.4 Relações entre Classes Quânticas

Nesta seção serão apresentas as relações de contingência formalmente provadas e definidas entre as classes de complexidade, apresentadas na seção anterior, e suas relações com as classes de complexidade clássicas, definidas no capítulo anterior.

3.4.1 Relações com a classe BQP

A relação mais simples envolvendo a classe BQP foi provada junto de sua definição, por Bernstein e Vazirani (1997) que mostraram que $BPP \subseteq BQP$, ou seja, computadores quânticos podem resolver em tempo polinomial qualquer problema clássico probabilístico de tempo polinomial, dado que as propriedades quânticas geram aleatoriedade.

Intuitivamente podemos pensar que é possível simular uma computação probabilística com circuitos quânticos, pois eles desempenham computações determinísticas eficientemente e podemos utilizar a porta de Hadamard para fazer $|0\rangle$ gerar $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ criando assim uma fonte aleatória de 0's e 1's.

A relação $BQP \subseteq PP$ é mais complexa de ser comprovada, como provada por Adleman, DeMarrais e Huang (1997). Porém segundo Watrous (2006) podemos intuiti-

vamente mostrar essa relação através de um algoritmo probabilístico clássico que simula um circuito quântico, essa simulação terá erro não limitado, ou seja, não será um valor constante menor que 1/2 mas como será usada apenas para mostrar a relação com a classe *PP*, que especifica erro menor que 1/2 (não necessariamente constante) ela é suficiente intuitivamente.

A demonstração simula a parte real e imaginária de um número complexo com dois *bits* e aplica transformações com portas universais quânticas utilizando circuitos probabilísticos, por exemplo, simulando a porta de Hadamard através de um arremesso de moeda[5]. O algoritmo por fim executa dois processos probabilísticos sobre uma entrada simultaneamente, se os dois processos derem o mesmo resultado então este resultado será aceito com probabilidade minimamente maior que 1/2, se os resultados forem diferentes então aleatoriamente escolhe aceita ou rejeita a entrada, o que é suficiente para pertencer a classe *PP*.

A relação entre *NP* e *BQP* ainda é desconhecida, apesar de que conjectura-se que problemas *NP-Completos* não estão em *BQP*, porém como $P \subseteq BQP$ a prova da relação $P \stackrel{?}{=} NP$ poderia dar uma luz nesta relação.

Dessa forma podemos acrescentar a classe *BQP* no diagrama de relações construído da seguinte forma:

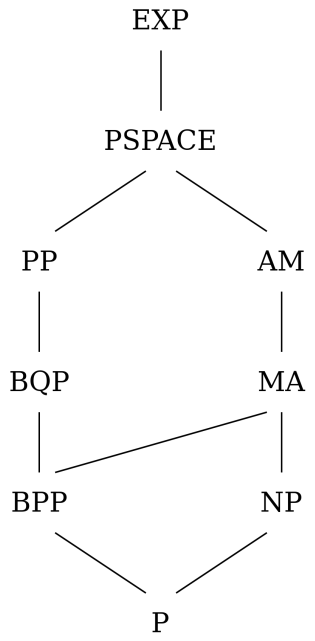

Figura 20 – Inclusão de BQP.

[5] Mais detalhes sobre a simulação podem ser vistos em <https://cs.uwaterloo.ca/~watrous/LectureNotes/CPSC519.Winter2006/22.pdf>.

3.4.2 Relações com a classe QMA

Similarmente a outras relações que já vimos anteriormente, a versão quântica da classe MA contém trivialmente sua versão clássica, pois podemos simular circuitos clássicos utilizando circuitos quânticos eficientemente, e com a porta de Hadamard podemos criar a aleatoriedade necessária para a computação clássica probabilística, então $MA \subseteq QMA$.

Análogo também a relação que a classe P tem com a classe NP, podemos fazer um paralelo entre a classe BQP que decide problemas em tempo polinomial com erro limitado em uma Máquina de Turing Quântica, com a classe QMA que verifica em tempo polinomial certificados usando uma Máquina de Turing Quântica. Como todo problema decidido em tempo polinomial, nessas condições, também é verificado em tempo polinomial então $BQP \subseteq QMA$.

Já a relação $QMA \subseteq PP$ não é trivial, provada por Watrous (2003) em conjunto com A. Kitaev e também demonstrada por Marriott e Watrous (2005), mostraram que o limite superior da classe BQP poderia ser generalizado para provar as mesmas inclusões para a classe QMA.

Apesar da relação entre MA e QMA ser trivial, a relação entre QMA e AM é desconhecida, as duas classes são conjecturadas serem incomparáveis, qualquer prova neste sentido seria um resultado enorme na complexidade quântica.

E com esse resultado, podemos atualizar o diagrama das relações de inclusão entre classes da seguinte forma:

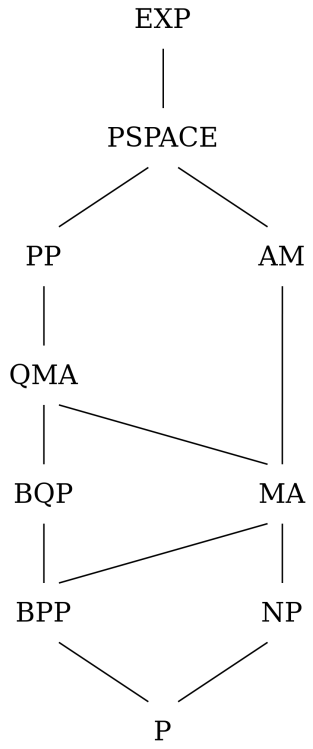

Figura 21 – Inclusão de QMA.

3.4.3 Relações com as classe QIP

Quase duas décadas após a prova de que $IP = PSPACE$, e uma década após $PSPACE \subseteq QIP$. No artigo preliminar por Jain et al. (2009) e publicado mais tarde em Jain et al. (2011), mostraram que $QIP \subseteq PSPACE$ aplicando de forma paralelizada o método de atualização de pesos multiplicativos de matriz a uma classe de programas semi definidos que captura o poder computacional de provas interativas quânticas, similar ao que foi feito por Jain, Upadhyay e Watrous (2009).

Provando que $QIP = IP = PSPACE$ mostramos que os sistemas quânticos de prova interativa não são mais poderosos do que os clássicos. Apesar que podem oferecer uma redução no número de mensagens requeridas. E que com 3 mensagens já se tem o poder equivalente a um número polinomial de mensagens trocadas.

A classe QAM é similar a classe $AM[2]$, em que o verificador Arthur gera uma sentença uniformemente aleatória e a envia ao provador Merlin, então Merlin responde com um certificado quântico limitado polinomialmente e Arthur pode verificar este certificado, da mesma forma que em AM, utilizando a sentença aleatória que gerou, porém usando uma Máquina de Turing Quântica. Assim notamos que $AM[2] \subseteq QAM$ é uma relação direta.

Como no protocolo $QIP[2]$ podemos realizar a primeira comunicação, do verificador ao provador, enviando uma sentença aleatória, como em QAM, mostramos diretamente

que $QAM \subseteq QIP[2]$, e por fim $AM[2] \subseteq QIP[2]$.

Apesar dos resultados encontrados é provável que no caso clássico, mesmo que $IP = PSPACE$, $PSPACE$ e AM sejam diferentes, porém nenhuma prova foi demonstrada ainda decidindo se AM e $PSPACE$ devem colapsar ou não na hierarquia de classes.

Dessa forma, podemos incluir $QIP[1]$, $QIP[2]$, e QIP na hierarquia de classes da seguinte forma:

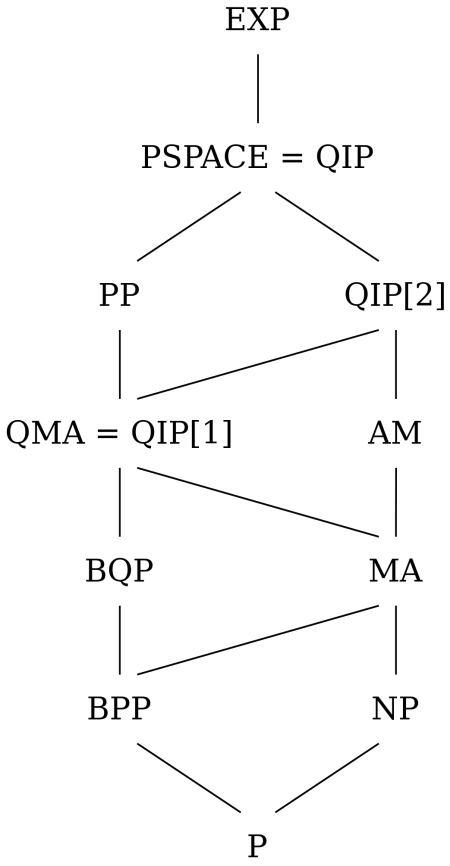

Figura 22 – Inclusão de QIP[1], QIP[2] e QIP.

4 Pontos em aberto e implicações

A quantidade de relações em aberto é significativa, dada a dificuldade de se provar verdadeira, ou não, a equivalência ou pertinência entre classes de complexidade. Devido a isso, neste capítulo serão apresentados os principais pontos em aberto no campo da complexidade computacional, enfatizando relações em aberto mais conhecidas ou que tendem a serem provadas em um futuro próximo, discutindo os impactos e implicações da prova destas relações, assim como quais são as conjecturas atuais e o que se espera que se prove verdadeiro no futuro.

A figura 22, construída no capítulo anterior, pode ser extremamente útil para visualizar e entender as relações e implicações que seguem.

4.1 P = NP

O problema se $P = NP$ é considerado por muitos o maior problema não resolvido na ciência da computação, segundo Hemaspaandra (2002) a maioria dos cientistas da computação acredita que $P \neq NP$, e a razão para isso é que depois de décadas de estudo sobre estes problemas não foi possível achar um algoritmo polinomial que decida qualquer um dos mais de 3000 problemas $NP - Completos$ conhecidos.

As implicações de $P = NP$ seriam gigantescas, tendo em vista que praticamente todas as áreas de estudo possuem problemas importantes $NP - Completos$, por exemplo, sistemas de segurança que se baseiam em algoritmos em NP estariam vulneráveis. A possibilidade de resolve-los em tempo polinomial transformaria o mundo completamente. Muitos problemas de otimização teriam sua solução ótima encontrada e a toda a classe de problemas que buscam soluções suficientemente boas em tempo polinomial perderia boa parte do seu valor. Segundo Sipser (1983) e Lautemann (1983) se $P = NP$ então $P = BPP$.

Poderíamos nos perguntar se o resultado que $P = NP$ implicaria que $BQP = P$, porém no momento ainda não sabemos, podem ainda existir problemas que estão em BQP que não estão em P. E mesmo se $BQP = P$, não implica que computadores quânticos não sejam úteis, de forma prática, poderíamos ter um ganho de desempenho expressivo na resolução de determinados problemas, aumentando a quantidade de instâncias de problemas que podemos resolver.

Já a prova que $P \neq NP$ não teria tantos benefícios práticos, porém ainda assim seria um grande avanço na teoria da complexidade computacional e poderia mudar o enfoque da pesquisa para outros problemas ainda em aberto.

4.2 P = PSPACE

Apesar de existir diversas outras classes entre a classe P e a classe $PSPACE$ no diagrama apresentado, não há qualquer prova que demonstre a igualdade ou desigualdade entre essas classes. De fato a única prova nesse sentido mostra que a classe P está estritamente contida na classe EXP, ou seja, $P \subsetneq EXP$, o que demonstra a dificuldade de encontramos provas que confirmem desigualdades entre as classes. Porém, este resultado também indica que pelo menos uma das inclusões entre P e EXP deve ser estrita.

Da mesma forma que conjectura-se que $P \neq NP$, conjectura-se que $NP \neq PSPACE$ e assim espera-se que $P \neq PSPACE$. Qualquer demonstração de $PSPACE = P$ levaria ao colapso da hierarquia de classes, que é improvável de acontecer.

4.3 P = BPP e NP = MA = AM

Sabemos que os problemas em P estão em BPP e existiam muitos problemas que estavam em BPP e não sabíamos se estavam também em P. Com o tempo o número destes problemas vem diminuindo. O problema de decidir se um determinado número é primo, por exemplo, foi por muito tempo, um problema famoso em BPP, porém Agrawal, Kayal e Saxena (2002) mostraram que há um algoritmo determinístico em tempo polinomial que o decide.

Ainda não sabemos se $P = BPP$, mas conjectura-se que sim, pois na maioria dos algoritmos aleatórios foi possível abrir mão da aleatoriedade e encontrar um algoritmo determinístico em P. Por razões similares conjectura-se que $NP = MA = AM$. Porém ainda não sabemos, por exemplo, se o problema do não isomorfismo em grafos, que está em AM, também está em MA ou NP.

4.4 Conjecturas sobre BQP

Com o início do estudo sobre computação quântica, a grande expectativa era ganharmos um aumento exponencial sobre o tempo utilizado em problemas considerados intratáveis, pois com o uso de *qubits* é possível codificar um conjunto maior de entradas, em um estado de superposição, e desta forma, realizar o processamento de todas as entradas em paralelo. Porém, como vimos, o processo de medição seleciona uma configuração dentre todos os estados em superposição com uma certa probabilidade.

Para que a computação quântica seja efetivamente mais eficiente que a computação clássica, é necessário que o algoritmo quântico explore o fenômeno de interferência quântica, de forma que, respostas inválidas interfiram destrutivamente[1] umas nas outras, visando

[1] Se um evento ocorrer com uma amplitude positiva e outra negativa.

diminuir a probabilidade de serem medidas, e respostas corretas interfiram construtivamente, aumentando a probabilidade de serem medidas. De fato, ainda só sabemos como fazer isso para alguns tipos especiais de problemas.

A relação entre BQP e NP é desconhecida e é um problema aberto até então, porém acredita-se que existem problemas em BQP que não estão em NP e problemas em NP que não estão em BQP, a evidência para isso baseia-se em oráculos[2].

Idealmente, gostaríamos de ter uma prova matemática que mostre que $NP \not\subset BQP$, mostrando que há problemas em NP que não estão em BQP, mas qualquer prova neste sentido implicaria que $P \neq NP$, pois $P \subseteq BQP$.

Dada a expectativa de solucionarmos problemas em NP em tempo polinomial em uma Máquina de Turing Quântica, poderíamos provar que $NP \subseteq BQP$ mostrando que há algum problema $NP - Completo$ que pertence também a BQP. O problema da fatoração de inteiros em seus fatores primos está em NP pois podemos verificar a fatoração simplesmente multiplicando os primos e comparando com o número inteiro, porém, até o momento este problema não foi provado ser $NP - Completo$ e nem estar em P.

Poderíamos imaginar que a prova $P \stackrel{?}{=} NP$ implicaria na prova que $BQP = QMA$ ou que $BQP \neq QMA$, ou vice-versa, no momento não há nenhum resultado formal que indique isso, porém qualquer avanço em alguma dessas questões poderia, possivelmente, desencadear avanço nas outras.

Atualmente também é conjecturado que há problemas em BQP, por exemplo, o Algoritmo de Shor (1997), que não estão em BPP, e dessa forma $BPP \subsetneq BQP$.

[2] Um oráculo é um dispositivo abstrato, que pode ser anexado a uma Máquina de Turing, e que decide problemas de decisão em uma única operação podendo ser útil para investigar relações entre classes.

5 Considerações Finais

Este trabalho apresentou um estudo sobre as principais classes de complexidade clássicas, probabilísticas e quânticas. Enfatizando além de suas definições formais e problemas contidos, as relações de equivalência entre as mesmas, provadas na literatura, criando um diagrama que possibilita uma visão geral do estado atual da hierarquia de classes de complexidade.

O presente trabalho também apresentou dois principais modelos de computação utilizados para definir classes de complexidade: o modelo abstrato de Máquinas de Turing e suas variações não-determinística, probabilística, quântica, e o modelo de Computação Quântica Circuital através da criação de circuitos com portas lógicas quânticas. Mostrando sempre que necessário conceitos fundamentais e exemplos do funcionamento, tanto dos modelos quânticos quanto dos modelos clássicos.

Por fim, este trabalho apresentou alguns dos principais pontos em aberto e conjecturas na área de teoria da complexidade, que merecem destaque, e que tem maior relevância na discussão sobre a capacidade da computação quântica frente a computação clássica.

Entende-se que ao estudar e mostrar as relações entre classes de complexidade, pode-se compreender melhor os limites dos modelos computacionais sobre aspectos de tempo e espaço, tornando possível a comparações entre eles. E dessa forma, alinhamos as expectativas sobre a Computação Quântica direcionando o esforço em pesquisa da melhor forma possível.

Espera-se que este trabalho possa ser útil em futuros trabalhos sobre classes de complexidade quânticas, servindo como uma base de conhecimento de conceitos fundamentais e fornecendo uma visão geral das relações entre as principais classes clássicas e quânticas.

Referências

ADLEMAN, L. M.; DEMARRAIS, J.; HUANG, M.-D. A. Quantum computability. *SIAM Journal on Computing*, v. 26, n. 5, p. 1524–1540, 1997. Disponível em: <https://doi.org/10.1137/S0097539795293639>.

AGRAWAL, M.; KAYAL, N.; SAXENA, N. Primes is in p. *Ann. of Math*, v. 2, p. 781–793, 2002.

AHARONOV, D.; NAVEH, T. Quantum np - a survey. *arXiv: Quantum Physics*, 2002.

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

BABAI, L. Trading group theory for randomness. In: ACM PRESS. *Proceedings Of The Seventeenth Annual Acm Symposium On Theory Of Computing - Stoc '85*. [S.l.], 1985. p. 421–429.

BABAI, L.; LUKS, E. M. Canonical labeling of graphs. In: *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*. New York, NY, USA: Association for Computing Machinery, 1983. (STOC '83), p. 171–183. ISBN 0897910990. Disponível em: <https://doi.org/10.1145/800061.808746>.

BABAI, L.; MORAN, S. Arthur-merlin games: A randomized proof system, and a hierarchy of complexity class. *J. Comput. Syst. Sci.*, Academic Press, Inc., USA, v. 36, n. 2, p. 254–276, abr. 1988. ISSN 0022-0000. Disponível em: <https://doi.org/10.1016/0022-0000(88)90028-1>.

BERNSTEIN, E.; VAZIRANI, U. Quantum complexity theory. *SIAM Journal on Computing*, v. 26, n. 5, p. 1411–1473, 1997. Disponível em: <https://doi.org/10.1137/S0097539796300921>.

BOOKATZ, A. D. Qma-complete problems. *Journal Quantum Information & Computation*, p. 361–383, 2014.

CHERNOFF, H. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, Institute of Mathematical Statistics, v. 23, n. 4, p. 493–507, 1952. ISSN 00034851. Disponível em: <http://www.jstor.org/stable/2236576>.

CHURCH, A. A note on the entscheidungsproblem. *Journal of Symbolic Logic*, v. 1, n. 1, p. 40–41, 1936.

COBHAM, A. The intrinsic computational difficulty of functions. In: *Proceedings Of The 1964 Congress On Logic, Mathematics And The Methodology Of Science*. [S.l.: s.n.], 1964. p. 24–30.

COOK, S. The complexity of theorem-proving procedures. In: *Proceedings of the Third Annual ACM Symposium on Theory of Computing*. [S.l.: s.n.], 1971. p. 151–158.

DEUTSCH, D. Quantum theory, the church-turing principle and the universal quantum computer. In: THE ROYAL SOCIETY. *Proceedings Of The Royal Society A : Mathematical, Physical and Engineering Sciences*. [S.l.], 1985. v. 400, p. 97–117.

DISCO, C.; MEULEN, B. V. D. *Getting new technologies together*: Studies in making sociotechnical order. [S.l.]: De Gruyter, 1998.

DU, D.; KO, K. *Theory of Computational Complexity*. [s.n.], 2014. (Wiley-Interscience series in discrete mathematics and optimization). ISBN 9781118595091. Disponível em: <https://books.google.com.br/books?id=E2sWogEACAAJ>.

EDMONDS, J. Paths, trees, and flowers. *Canadian Journal Of Mathematics*, v. 17, p. 449–467, 1965.

GILL, J. T. Computational complexity of probabilistic turing machines. *SIAM Journal on Computing*, v. 4, n. 6, p. 675–695, 1977.

GOLDREICH, O. *Introduction to Complexity Theory - Lecture Notes*. 1999.

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*, Association for Computing Machinery, New York, NY, USA, v. 38, n. 3, p. 690–728, jul. 1991. ISSN 0004-5411. Disponível em: <https://doi.org/10.1145/116825.116852>.

GOLDWASSER, S.; MICALI, S.; RACKOFF, C. The knowledge complexity of interactive proof-systems. In: *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*. New York, NY, USA: Association for Computing Machinery, 1985. (STOC '85), p. 291–304. ISBN 0897911512. Disponível em: <https://doi.org/10.1145/22145.22178>.

GOLDWASSER, S.; SIPSER, M. Private coins versus public coins in interactive proof systems. In: *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*. New York, NY, USA: Association for Computing Machinery, 1986. (STOC '86), p. 59–68. ISBN 0897911938. Disponível em: <https://doi.org/10.1145/12130.12137>.

HEMASPAANDRA, L. A. Sigact news complexity theory column 36. *SIGACT News*, Association for Computing Machinery, New York, NY, USA, v. 33, n. 2, p. 34–47, jun. 2002. ISSN 0163-5700. Disponível em: <https://doi.org/10.1145/564585.564599>.

HOPCROFT, J. E.; MOTWANI, R.; ULLMAN, J. D. *Introduction to Automata Theory, Languages, and Computation*. [S.l.]: Addison-Wesley, 2001.

JAIN, R. et al. *QIP = PSPACE*. 2009.

JAIN, R. et al. Qip = pspace. *J. ACM*, Association for Computing Machinery, New York, NY, USA, v. 58, n. 6, dez. 2011. ISSN 0004-5411. Disponível em: <https://doi.org/10.1145/2049697.2049704>.

JAIN, R.; UPADHYAY, S.; WATROUS, J. *Two-message quantum interactive proofs are in PSPACE*. 2009.

KITAEV, A.; WATROUS, J. Parallelization, amplification, and exponential time simulation of quantum interactive proof systems. In: *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*. New York, NY, USA: Association for Computing Machinery, 2000. (STOC '00), p. 608–617. ISBN 1581131844. Disponível em: <https://doi.org/10.1145/335305.335387>.

KRUSKAL, J. B. On the shortest spanning subtree of a graph and the traveling salesman problem. In: AMERICAN MATHEMATICAL SOCIETY. *Proceedings of the American Mathematical Society*. [S.l.], 1956. p. 48–50.

LAUTEMANN, C. Bpp and the polynomial hierarchy. *Information Processing Letters*, v. 17, n. 4, p. 215 – 217, 1983. ISSN 0020-0190. Disponível em: <http://www.sciencedirect.com/science/article/pii/0020019083900443>.

LEVIN, L. A. Universal sequential search problems. *Probl. Peredachi Inf.*, v. 9, p. 115–116, 1973.

MARRIOTT, C.; WATROUS, J. Quantum arthur-merlin games. *CoRR*, abs/cs/0506068, 2005. Disponível em: <http://arxiv.org/abs/cs/0506068>.

MELO, B. L. M. d.; CHRISTOFOLETTI, T. V. D. Computação quântica : Estado da arte. p. 0–4, 2003.

NIELSEN, M. A.; CHUANG, I. *Quantum computation and quantum information*. [S.l.]: Cambridge University Press, 2010.

NISHIMURA, H.; OZAWA, M. *Computational complexity of uniform quantum circuit families and quantum Turing machines*. 1999.

OZAWA, M. On the halting problem for quantum turing machines. 01 1998.

PAPADIMITRIOU, C. H. *Computational complexity*. [S.l.]: Addison-Wesley, 1994. I-XV, 1-523 p. ISBN 978-0-201-53082-7.

POLLACHINI, G. G. *Computacão Quântica: Uma abordagem para estudantes de graduação em Ciências Exatas*. 2018.

Prim, R. C. Shortest Connection Networks And Some Generalizations. *Bell System Technical Journal*, v. 36, n. 6, p. 1389–1401, nov. 1957.

RABIN, M. O. *Degree of difficulty of computing a function, and a partial ordering of recursive sets*. [S.l.], 1960.

RIVEST, R. L.; SHAMIR, A.; ADLEMAN, L. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, Association for Computing Machinery, New York, NY, USA, v. 21, n. 2, p. 120–126, fev. 1978. ISSN 0001-0782. Disponível em: <https://doi.org/10.1145/359340.359342>.

Ruzzo, W. L. On uniform circuit complexity. In: *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*. [S.l.: s.n.], 1979. p. 312–318.

SAVITCH, W. J. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, Academic Press, Inc., USA, v. 4, n. 2, p. 177–192, abr. 1970. ISSN 0022-0000. Disponível em: <https://doi.org/10.1016/S0022-0000(70)80006-X>.

Shamir, A. Ip=pspace (interactive proof=polynomial space). In: *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*. [S.l.: s.n.], 1990. p. 11–15 vol.1.

SHOR, P. W. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *Siam Journal On Computing*, v. 26, n. 5, p. 1484–1509, 1997.

SIPSER, M. A complexity theoretic approach to randomness. In: *PROCEEDINGS OF THE 15TH ANNUAL ACM SYMPOSIUM ON THEORY OF COMPUTING*. [S.l.: s.n.], 1983. p. 330–335.

SIPSER, M. *Introduction on the Theory of Computation*. 2. ed. [S.l.]: Course Technology Cengage Lerning, 2006.

TURING, A. M. On computable numbers, with an application to the entscheidungsproblem. In: WILEY. *Proceedings Of The London Mathematical Society*. [S.l.], 1937. p. 230–265.

VERESCHCHAGIN, N. K. On the power of pp. In: IEEE. *Proceedings of the Seventh Annual Structure in Complexity Theory Conference*. [S.l.], 1992. p. 138–143.

WATROUS, J. *PSPACE has 2-round quantum interactive proof systems*. 1999.

WATROUS, J. Succinct quantum proofs for properties of finite groups. In: *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*. USA: IEEE Computer Society, 2000. (FOCS '00), p. 537. ISBN 0769508502.

WATROUS, J. Pspace has constant-round quantum interactive proof systems. *Theoretical Computer Science*, v. 292, n. 3, p. 575 – 588, 2003. ISSN 0304-3975. Algorithms in Quantum Information Prcoessing. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0304397501003759>.

WATROUS, J. *Lecture 22: Quantum computational complexity*. [S.l.]: UniversityofCalgary, 2006.

YAO, A. C.-C. Quantum circuit complexity. In: *Proceedings of the 1993 IEEE 34th Annual Foundations of Computer Science*. USA: IEEE Computer Society, 1993. (SFCS '93), p. 352–361. ISBN 0818643706. Disponível em: <https://doi.org/10.1109/SFCS.1993.366852>.

APÊNDICE A – Artigo

Modelos Computação e Classes de Complexidade Quânticas Relações com a Computação Clássica

Maurício M. Barbosa[1]

[1]Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)

mauriciomb97@gmail.com

Abstract. *Quantum computers are increasingly less distant reality and, with them, comes a great expectation of improving the efficiency of algorithms that seek to solve different types of problems considered classically intractable. In this sense, a study on theoretical models of computation and the main classical and quantum complexity classes will be done, showing contingency and equivalence relations between them, in order to define the limits of quantum computing on theoretical aspects of computability and treatability. Also defining the main open relationships and conjectures in the literature on the theory of quantum computational complexity.*

Resumo. *Computadores quânticos são cada vez uma realidade menos distante e, com eles, vem a grande expectativa de melhorar a eficiência de algoritmos que buscam resolver diversos tipos de problemas considerados intratáveis classicamente. Neste sentido, será feito neste trabalho um estudo sobre modelos teóricos de computação e as principais classes de complexidade clássicas e quânticas, mostrando relações de contingência e equivalência entre elas, de forma a definir os limites da computação quântica sobre aspectos teóricos de computabilidade e tratabilidade. Definindo também as principais relações em aberto e conjecturas na literatura sobre a teoria da complexidade computacional quântica.*

1. Introdução

Até meados de 1965 não havia nenhuma previsão real sobre o futuro do hardware quando Gordon E. Moore fez sua profecia, na qual o número de transistores dos chips teria um aumento de 100%, pelo mesmo custo, a cada período de 18 meses. Essa profecia tornou-se realidade e acabou ganhando o nome de Lei de Moore [Disco and Van Der Meulen 1998].

Segundo Nielsen e Chuang [Nielsen and Chuang 2010], à medida que a escala dos transistores diminuía, ela se aproximava cada vez mais de limites físicos fundamentais. Nos dias atuais, empresas como Intel e Nvidia apostam em outras alternativas aos chips de silício, investindo cada vez mais na Computação Quântica e também em Computação Biológica, para superar este obstáculo.

A Computação Quântica é um novo paradigma de computação em que se utilizam sistemas quânticos para se realizar processamento de informação. E nela são introduzidos recursos não existentes na Computação Clássica. [Pollachini 2018].

Segundo Melo e Christofoletti [Melo and Christofoletti 2003], a computação quântica é uma proposta para realizar o processo da computação usando álgebra quântica.

A mecânica quântica é fundamental nesse processo. O computador clássico opera com uma sequência de zeros e uns, de modo que qualquer ação computacional pode ser traduzida em última instância por uma sequência desses algarismos. E esses zeros e uns, são na verdade estados lógicos de transistores. Não é possível ter os dois ao mesmo tempo.

Já os computadores quânticos mantém um conjunto de qubits. Um qubit pode conter um 1, um 0 ou uma sobreposição destes. Em outras palavras, pode conter tanto um 1 como um 0 ao mesmo tempo. O computador quântico funciona pela manipulação destes qubits. Com isso, uma coleção de qubits poderia representar uma fileira de números ao mesmo tempo, e um computador quântico poderia processar toda uma entrada de dados simultaneamente [Melo and Christofoletti 2003].

Com isso, surgem as questões sobre o poder computacional de um computador quântico, que motivam a produção deste trabalho. Seriam computadores quânticos capazes de solucionar problemas que não podem ser solucionados por um modelo clássico de computação, como uma máquina de Turing? Seriam os computadores quânticos capazes de tratar problemas que hoje são considerados intratáveis nos modelos de computação clássicos? Quais as classes de problemas que podem ser efetivamente resolvidos com a utilização de algoritmos quânticos? Quais são as relações entre as classes de complexidade clássicas como por exemplo, P e NP, com as classes de complexidade quânticas?

Segundo Pollachini [Pollachini 2018], a expectativa é que um modelo híbrido entre Computação Clássica e Quântica seja a tecnologia emergente no cenário atual; A computação seria realizada em um processador quântico para problemas nos quais há vantagens no uso da Computação Quântica, como espera-se que ocorra com os problemas chamados NP-difíceis. Dentre as principais aplicações previstas para a Computação Quântica, destaca-se resolução de problemas de otimização, *machine learning*, desenvolvimento de novos materiais, fármacos e processos químicos.

O algoritmo quântico de Shor [Shor 1997], possibilita que fatorações de números primos e o cálculo de logaritmos discretos sejam feitos de forma eficiente, impactando assim sistemas de criptografias que se baseiam nessa dificuldade prática, como por exemplo, o algoritmo de criptografia assimétrica RSA [Rivest et al. 1978] e amplamente ainda utilizado até hoje.

Dadas as aplicações futuras e expectativas desse novo paradigma surge a necessidade de pesquisa em ciência de base, preocupada em responder como as coisas se relacionam e quais são as potencialidades da computação quântica, no caso, como se organizam as classes de complexidade quânticas e quais são suas relações com as outras classes clássicas.

O objetivo geral deste trabalho é apresentar o modelo de Máquina de Turing Quântica e seu funcionamento estabelecendo relações entre as classes quânticas de complexidade e as classes clássicas, enfatizando os principais resultados, pontos em aberto e implicações.

Este trabalho será apresentado da seguinte forma: na seção 2 será apresentado o modelo de Máquina de Turing Quântica, na seção 3 serão apresentadas, de modo incremental, as principais classes de complexidades clássicas e quânticas e suas relações, já na seção 4 serão mostrados problemas em aberto, implicações e no capítulo 5 as considerações finais do trabalho.

2. Máquina de Turing Quântica

Uma Máquina de Turing Quântica, definida por Deutsch [Deutsch 1985], diferente das outras variantes de Máquinas de Turing pode encontrar-se numa superposição de configurações a cada instante, sendo a superposição de configurações uma combinação linear de configurações. Em uma Máquina de Turing Quântica é possível, a cada passo de computação, efetuar múltiplas transições nas diversas configurações. Potencialmente essa característica pode ser explorada para se obter algoritmos quânticos eficientes para problemas hoje considerados difíceis.

Ela também difere no sentido em que, ao final de sua execução, a máquina realiza uma medição, que consiste em escolher aleatoriamente uma das configurações e devido a essa característica probabilística, essa máquina pode produzir diferentes saídas para uma mesma configuração de entrada.

Segundo Ozawa [Ozawa 1998], uma Máquina de Turing Quântica $\mathcal{Q}$, é um sistema quântico consistido de um processador, uma fita infinita bilateralmente e um cabeçote para ler e escrever símbolos na fita. Sua configuração é determinada pela configuração q do processador, de um conjunto finito Q de estados, pela configuração da fita T representada por uma cadeia infinita de um conjunto finito Σ de símbolos, e a posição discreta do cabeçote ξ, sendo um valor no conjunto $\mathbb{Z}$ de inteiros.

Desta forma, qualquer configuração C de $\mathcal{Q}$ pode ser representada por uma tripla $C = (q, T, \xi)$ na configuração de espaço $Q \times \Sigma^* \times \mathbb{Z}$, sendo q o estado atual, T a configuração atual da fita e ξ a posição discreta do cabeçote sobre a fita. Assim o estado de $\mathcal{Q}$ é representado por um vetor unitário no espaço de Hilbert $\mathcal{H}$ gerado pela configuração de espaço. A base computacional é representada por $|C\rangle = |q\rangle\,|T\rangle\,|\xi\rangle$ para qualquer configuração $C = (q, T, \xi) \in Q \times \Sigma^* \times \mathbb{Z}$ podendo escrever também como: $|q, T, \xi\rangle = |q\rangle\,|T\rangle\,|\xi\rangle$.

A função de transição δ, pode ser definida, como por Bernstein e Vazirani [Bernstein and Vazirani 1997], com $\tilde{C}$ sendo um conjunto de amplitudes $\alpha \in \mathbb{C}$, no conjunto dos números complexos, no qual:

$$\delta : Q \times \Sigma \to \tilde{C}^{Q \times \Sigma \times \{E,D\}}$$

Dessa forma, dado um estado e um símbolo na fita, a função faz um mapeamento para um conjunto $\tilde{C}$. Cada elemento neste conjunto identifica uma amplitude e atrelada a cada amplitude está o próximo estado da máquina em Q, o símbolo em Σ que deve ser escrito na fita e define a direção do movimento do cabeçote podendo mover-se à esquerda (E) ou à direita (D). A função restringe que a soma dos quadrados das amplitudes seja igual a 1.

A Máquina de Turing Quântica define um operador linear, chamado de operador de evolução temporal, se a Máquina $\mathcal{Q}$ começa numa configuração C com estado corrente q e símbolo a na fita, então após um passo $\mathcal{Q}$ vai estar em uma superposição de configurações, no estado $\psi = \sum_i \alpha_i C_i$, onde cada valor de amplitude $\alpha_i \neq 0$ foi definido na função de transição $\delta(q, a)$, sendo C_i a configuração da máquina após o processador ir para o estado definido na função de transição, escrevendo na fita onde o cabeçote se encontra e movendo o cabeçote em uma direção especificada. Estendendo esse mapeamento

para o espaço inteiro temos o operador linear de evolução temporal U.

A computação consiste em preparar primeiramente o estado inicial $|C_0\rangle$ de tal forma que $|C_0\rangle = |q_0\rangle\,|T_{in}\rangle\,|0\rangle$, neste caso T_{in} é a entrada da máquina codificada e transformada em uma cadeia de entrada na fita, então a computação procede, chamada também de evolução do sistema, transformando o estado inicial no estado final. A dinâmica de $\mathcal{Q}$ é descrita por um operador unitário U em $\mathcal{H}$, que especifica a evolução de qualquer estado $|\psi(t)\rangle$ durante uma única etapa computacional no tempo t.

O operador de evolução age no estado $|\psi\rangle$ em t_0 e nos dá o estado $|\psi\rangle$ no tempo t:

$$|\psi(t)\rangle = U^t\,|\psi(0)\rangle$$

Ainda segundo Ozawa [Ozawa 1998], o resultado de uma computação é obtido fazendo uma medição na fita após a computação ter finalizado, diferente do caso clássico, a configuração da máquina não pode ser monitorada durante a computação por causa do efeito causado por uma medição.

2.1. Exemplo de funcionamento

Nesta seção iremos exemplificar o funcionamento de uma Máquina de Turing Quântica que reconhece a linguagem regular 0^+1^+, como a expressão regular indica, uma cadeia pertence a essa linguagem se começar com uma sequência de um ou mais 0's, e em seguida, uma sequência de um ou mais 1's. Essa linguagem pode ser reconhecida, sem dificuldades, por um autômato finito, mas por questões de simplicidade será utilizada neste contexto.

Para demonstrar o funcionamento utilizaremos uma notação simplificada, que tem a forma similar à $|\#000q_i11\#\rangle$, neste caso ela representa que o conteúdo da fita é $|\#00011\#\rangle$, o estado atual da máquina é $|q_i\rangle$, e o cabeçote se localiza na posição subsequente à posição do estado, ou seja, neste caso o cabeçote está sobre o primeiro valor 1 na fita. Adicionaremos o símbolo $\#$ para indicar o começo e o fim da cadeia de entrada. Também usaremos o símbolo $\vdash$ para indicar um passo de computação na máquina, no lado esquerdo, a configuração antes do passo, e no lado direito, a configuração após.

Podemos definir uma Máquina de Turing Quântica que reconhece 0^+1^+, da seguinte forma:

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_a, q_r\}$, sendo q_0 o estado inicial, q_a o estado de aceitação e q_r o estado de rejeição;

$\Sigma = \{0, 1\}$;

E com a função de transição definida na Tabela 1. Esta função leva em consideração que a entrada será pré-processada, adicionando o símbolo $\#$ no começo e fim da entrada, e o cabeçote da Máquina será posicionado no meio da cadeia de entrada, como a fita é infinita bilateralmente, podemos, de forma equivalente, configurar a fita para que o centro da cadeia esteja na posição 0 na configuração inicial.

δ	0	1	#
$\rightarrow q_0$	$\frac{1}{\sqrt{2}}^{(q_1,0,E)}$, $\frac{1}{\sqrt{2}}^{(q_2,0,D)}$	$\frac{1}{\sqrt{2}}^{(q_4,1,E)}$, $\frac{1}{\sqrt{2}}^{(q_5,1,D)}$	$1^{(q_r,0,E)}$
q_1	$1^{(q_1,0,E)}$	$1^{(q_r,1,E)}$	$1^{(q_a,0,D)}$
q_2	$1^{(q_2,0,D)}$	$1^{(q_3,1,D)}$	$1^{(q_r,0,E)}$
q_3	$1^{(q_r,0,D)}$	$1^{(q_3,1,D)}$	$1^{(q_a,0,E)}$
q_4	$1^{(q_6,0,E)}$	$1^{(q_4,1,E)}$	$1^{(q_r,0,D)}$
q_5	$1^{(q_r,0,D)}$	$1^{(q_5,1,D)}$	$1^{(q_a,0,E)}$
q_6	$1^{(q_6,0,E)}$	$1^{(q_r,1,E)}$	$1^{(q_a,0,D)}$

Tabela 1. Função de transição da Máquina de Turing Quântica

Para exemplificar faremos uma simulação com a entrada $|00011\rangle$ que pertence à linguagem. Pré-processamos a entradas, como descrito anteriormente, e a configuração inicial da Máquina no cenário será: $|\#00q_0011\#\rangle$.

A função de transição indica que $\delta(q_0, 0) = \frac{1}{\sqrt{2}}^{(q_1,0,E)}$, $\frac{1}{\sqrt{2}}^{(q_2,0,D)}$. Então, após um passo de computação teremos:

$$|\#00q_0011\#\rangle \vdash \frac{1}{\sqrt{2}} |\#0q_10011\#\rangle + \frac{1}{\sqrt{2}} |\#000q_211\#\rangle$$

Podemos ver que a máquina está em uma superposição de estados e em um próximo passo de computação, avaliaremos a função de transição tanto para $\delta(q_1, 0)$ quanto para $\delta(q_2, 1)$, o que resulta em:

$$\frac{1}{\sqrt{2}} |\#0q_10011\#\rangle + \frac{1}{\sqrt{2}} |\#000q_211\#\rangle \vdash \frac{1}{\sqrt{2}} \times 1 |\#q_100011\#\rangle + \frac{1}{\sqrt{2}} \times 1 |\#0001q_31\#\rangle$$

No próximo passo de computação, avaliaremos para $\delta(q_1, 0)$ e $\delta(q_3, 1)$, resultando em:

$$\frac{1}{\sqrt{2}} |\#q_100011\#\rangle + \frac{1}{\sqrt{2}} |\#0001q_31\#\rangle \vdash \frac{1}{\sqrt{2}} \times 1 |q_1\#00011\#\rangle + \frac{1}{\sqrt{2}} \times 1 |\#00011q_3\#\rangle$$

Por fim, será avaliada a função de transição em $\delta(q_1, \#)$ e $\delta(q_3, \#)$, que resulta em:

$$\frac{1}{\sqrt{2}} |q_1\#00011\#\rangle + \frac{1}{\sqrt{2}} |\#00011q_3\#\rangle \vdash \frac{1}{\sqrt{2}} \times 1 |0q_a00011\#\rangle + \frac{1}{\sqrt{2}} \times 1 |\#0001q_a10\rangle$$

Todos os ramos chegaram em um estado final, finalizando o processamento na máquina. Podemos efetuar uma medição no estado atual da máquina para saber se ela finalizou em um estado de aceitação ou rejeição. No caso desta sentença há $\frac{1}{2}$ de chance de obtermos $|q_a\rangle |000011\#\rangle$ e $\frac{1}{2}$ de obtermos $|q_a\rangle |\#000110\rangle$. Como, neste caso, o valor da fita é irrelevante para a tomada de decisão, aceitamos a entrada com probabilidade 1.

3. Classes de complexidade e relações

A classe P, também conhecida como *Polynomial-Time* é uma das classes de complexidade fundamentais, definida inicialmente por Edmonds, Cobham e Rabin [Edmonds 1965] [Cobham 1964] [Rabin 1960]. A classe P contém todos os problemas de decisão que podem ser decididos por uma Máquina de Turing Determinística usando uma quantidade polinomial de tempo de computação, ou seja existe um polinômio $p(n)$, sendo n o comprimento da entrada, em que a máquina sempre para e decide após $p(n)$ passos.

Com as contribuições de Cobham e Edmonds [Cobham 1964][Edmonds 1965] podemos assumir que os problemas serão considerados tratáveis, ou seja, que podem ser viáveis e eficientes de serem decididos em algum dispositivo computacional, somente se podem ser computados em tempo polinomial por um Máquina de Turing Determinística.

A classe NP, também conhecida por *Non-Deterministic Polynomial Time*, pode ser definida formalmente, como feito por Sipser [Sipser 2006], sendo a classe de linguagens que uma Máquina de Turing Não-determinística consegue decidir em tempo polinomial ao tamanho da entrada, consideramos NP uma classe de problemas para os quais não temos soluções eficientes e consideradas tratáveis.

Segundo Sipser [Sipser 2006] a classe NP também pode ser definida de forma equivalente como sendo a classe de linguagens que pode ser verificada em tempo polinomial. Ou seja, dada uma cadeia de entrada é possível verificar se ela pertence à linguagem ou não em tempo polinomial. Sipser define formalmente um Verificador, para uma linguagem L, sendo um algoritmo V, onde:

$$L = \{w \mid V \; aceita \; < w, c > \; para \; alguma \; cadeia \; c\}$$

sendo c um certificado ou prova que pode ser usado para verificar que uma cadeia w pertence a linguagem L. Se uma linguagem L possui um verificador que roda em tempo polinomial então $L \in NP$.

O trabalho de Cook e Levin [Cook 1971] e [Levin 1973] mostrou que existem problemas em NP cuja complexidade é a mesma da classe inteira. Os chamados problemas $NP - Completos$ são considerados os problemas mais difíceis da classe; A implicação disso é que se existir, para qualquer problema $NP - Completo$, um algoritmo que o solucione em tempo polinomial, todos os outros problemas em NP também o serão.

Por definição podemos afirmar que $P \subseteq NP$, dado que, todo problema que pode ser resolvido em tempo polinomial deterministicamente, também é resolvido em tempo polinomial por uma Máquina de Turing não-determinística que não precisa utilizar o não-determinismo.

Definida inicialmente por Gill [Gill 1977] a classe *Bounded-Error Probabilistic Polynomial Time* - BPP é a classe em que todos os problemas de decisão são solucionados em tempo polinomial com erro máximo de $1/3$. Ou seja, uma cadeia de entrada que pertence a linguagem tem no mínimo probabilidade $2/3$ de ser aceita pela máquina, e se a cadeia não pertence a linguagem então a máquina a rejeita com probabilidade pelo menos $2/3$.

Pode-se notar que todos os problemas em P estão contidos em BPP, já que o erro máximo de uma Máquina de Turing Determinística é 0. Já o relacionamento entre

NP e BPP é desconhecido, não podemos afirmar que $BPP \subseteq NP$ ou $NP \subseteq BPP$.

A classe MA é a versão probabilística da classe NP. Segundo Babai [Babai 1985], MA é um conjunto de linguagens L contidas em $\{0,1\}$* onde, para cada uma delas, existe uma Máquina de Turing M, probabilística, que computa em tempo polinomial para todas as entradas w. Caso w pertença a linguagem então existe um certificado c que $M(w,c)$ aceita com probabilidade no mínimo $2/3$. Caso w não pertença a linguagem então para qualquer certificado c, $M(w,c)$ aceita com probabilidade máxima $1/3$.

Desta definição nota-se que $BPP \subseteq MA$, dado que um problema decidido em uma Máquina de Turing Probabilística com erro máximo de $1/3$ também é verificado com erro máximo $1/3$. É imediato que $NP \subseteq MA$, tendo em vista que, NP verifica deterministicamente um certificado em tempo polinomial.

A classe AM (Arthur-Merlin) ou $AM[2]$, introduzida por Babai [Babai 1985] é composta por problemas de decisão que podem ser verificados em tempo polinomial por um protocolo de troca de mensagens entre um Verificador e um Provador com duas mensagens. Ou seja há apenas um conjunto de consulta e resposta. O Verificador, joga algumas moedas aleatórias e envia o resultado dos seus lançamentos para o Provador que responde com um certificado. O Verificador então decide se aceita ou rejeita o certificado em tempo polinomial e com erro máximo $1/3$ utilizando as moedas sorteadas.

A classe MA está contida em $AM[2]$, tendo em vista que o protocolo de troca de mensagens entre Provador e Verificador só necessita trocar uma mensagem em MA. A classe $AM[k]$, definida similarmente a classe AM, porém com k rodadas de interação, é equivalente a classe $AM[2]$ para todo $k > 2$ [Babai and Moran 1988].

Definida por Gill [Gill 1977], a classe PP (*Probabilistic Polynomial-Time*) é a classe dos problemas de decisão que são decididos por uma Máquina de Turing em tempo polinomial ao tamanho da entrada com erro menor que $1/2$ para todas as respostas. Pode-se mostrar que a classe $BPP \subseteq PP$, dado que os problemas da classe BPP tem erro menor que $1/3$ para todas as instâncias, e a classe PP, como definida, contém todos os problemas com erro menor que $1/2$. Mais importante que a relação acima e incluindo a mesma, Vereschchagin mostrou que $MA \subseteq PP$ [Vereschchagin 1992].

Segundo Sipser [Sipser 2006], $PSPACE$ é a classe dos problemas de decisão que utilizam espaço polinomial em uma Máquina de Turing Determinística. A relação $PP \subseteq PSPACE$ não é tão fácil de ser visualizada, uma maneira de entender essa relação, é notar que um algoritmo probabilístico que roda em tempo polinomial em uma Máquina de Turing Não-determinística, só necessita de, no máximo, espaço polinomial para cada ramo de computação.

Como resultado de Shamir [Shamir 1990], a classe dos sistemas de provas interativas $IP[poly]$, com um número polinomial de rodadas de interação ao tamanho da entrada k, foi provada ser equivalente a classe $PSPACE$. E utilizando o resultado de Goldwasser e Sipser [Goldwasser and Sipser 1986] que $IP[k] \subseteq AM[k+2]$, mostramos que $IP[poly] = AM[poly] = PSPACE$, como $AM[k] \subseteq AM[poly]$ então $AM \subseteq PSPACE$.

Segundo Papadimitriou [Papadimitriou 1994] a classe $EXPTIME$, ou simples-

mente EXP, pode ser definida como o conjunto de todos os problemas de decisão que podem ser resolvidos por uma Máquina de Turing Determinística em tempo exponencial ao tamanho n da entrada para qualquer $k \in \mathbb{N}$.

Podemos provar formalmente, como mostrado por Sipser [Sipser 2006, 308]: Para $f(n) \geq n$, uma Máquina de Turing que usa $f(n)$ espaço pode ter no máximo $f(n)2^{O(f(n))}$ diferentes configurações. A computação por uma Máquina de Turing que decide não deve repetir uma configuração. Portanto uma Máquina de Turing que usa espaço $f(n)$ deve computar em tempo $f(n)2^{f(n)}$, então $PSPACE \subseteq EXPTIME$.

A relação entre as classes de complexidade descritas podem ser resumidas na Figura 1

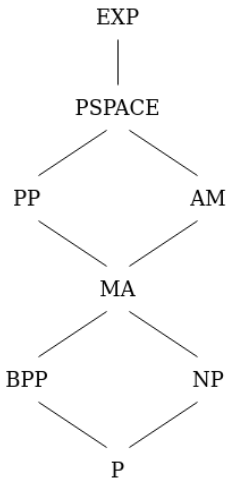

Figura 1. Relações entre Classes de Complexidade Clássicas.

Segundo Bernstein e Vazirani [Bernstein and Vazirani 1997], a classes BQP *Bounded-error Quantum Polynomial-time* é a classe dos problemas (linguagens) de decisão que podem ser resolvidos em tempo polinomial por uma Máquina de Turing Quântica com erro probabilístico limitado a $1/3$, para todas as cadeias de entrada. Geralmente é identificada como sendo a classe dos problemas tratáveis para computadores quânticos.

Bernstein e Vazinari [Bernstein and Vazirani 1997] mostraram que $BPP \subseteq BQP$, ou seja, computadores quânticos podem resolver em tempo polinomial qualquer problema clássico probabilístico de tempo polinomial, dado que as propriedades quânticas geram aleatoriedade. É possível provar também a relação $BQP \subseteq PP$ [Adleman et al. 1997].

Análogo a forma que a classe MA se relaciona com a classe BPP e a classe NP se relaciona com a classe P, a classe QMA se relaciona com a classe BQP. Essa classe, definida por [Watrous 2000], contém todos os problemas de decisão que são verificados por uma Máquina de Turing Quântica em tempo polinomial com probabilidade de erro menor que $1/3$. Basicamente a classe QMA expande a definição de NP para permitir que a verificação seja realizada por uma computação quântica e o certificado possa ser um estado quântico.

Similarmente a outras relações que já vimos anteriormente, a versão quântica da classe *MA* contém trivialmente sua versão clássica, pois podemos simular circuitos clássicos utilizando circuitos quânticos eficientemente, e podemos criar a aleatoriedade necessária para a computação clássica probabilística, então $MA \subseteq QMA$.

Análogo também a relação que a classe *P* tem com a classe *NP*, podemos fazer um paralelo entre a classe *BQP* que decide problemas em tempo polinomial com erro limitado em uma Máquina de Turing Quântica, com a classe *QMA* que verifica em tempo polinomial certificados usando uma Máquina de Turing Quântica. Como todo problema decidido em tempo polinomial, nessas condições, também é verificado em tempo polinomial então $BQP \subseteq QMA$.

Já a relação $QMA \subseteq PP$ não é trivial, provada por Watrous [Watrous 2003] em conjunto com A. Kitaev e também demonstrada por Marriott [Marriott and Watrous 2005], mostraram que o limite superior da classe *BQP* poderia ser generalizado para provar as mesmas inclusões para a classe *QMA*.

A classe QIP (Quantum Interactive Polynomial time), introduzida e definida formalmente por Watrous [Watrous 1999]. Pode ser informalmente descrita como sendo o conjunto de problemas de decisão, que podem ser verificados por um protocolo quântico de provas interativas. Nesta classe o Verificador deve verificar através de algoritmos quânticos e em tempo polinomial, enquanto o Provador tem recursos computacionais ilimitados. Neste protocolo podem ser trocadas um número polinomial de mensagens que podem ser estados quânticos.

Kitaev e Watrous [Kitaev and Watrous 2000] provaram que qualquer sistema quântico de provas interativas com um número polinomial de rodadas e com erro limitado bilateral pode ser paralelizado a um sistema de prova interativa quântica com erro unilateral exponencialmente pequeno no qual o provador e o verificador trocam apenas 3 mensagens. Dessa forma provando que: $\forall k > 3, QIP[k] = QIP[3] = QIP$.

Uma década após a prova que $PSPACE \subseteq QIP[3]$ [Watrous 2003]. Foi mostrado que $QIP \subseteq PSPACE$ aplicando de forma paralelizada o método de atualização de pesos multiplicativos de matriz a uma classe de programas semi definidos que captura o poder computacional de provas interativas quânticas [Jain et al. 2011], provando que $QIP[3] = PSPACE$.

A classe $QIP[2]$, poderá trocar no máximo duas mensagens entre o Provador e o Verificador. Esta classe está trivialmente contida em $QIP[3]$ e contém trivialmente a classe $QIP[1]$.

A classe $QIP[1]$ é equivalente a classe QMA por definição, dado que somente uma mensagem pode ser enviada, do Provador para o Verificador e será verificada por uma Máquina de Turing Quântica em tempo polinomial com erro máximo de $1/3$.

Com os resultados mostrados, podemos acrescentar as classes de complexidade quânticas no diagrama, como na Figura 2.

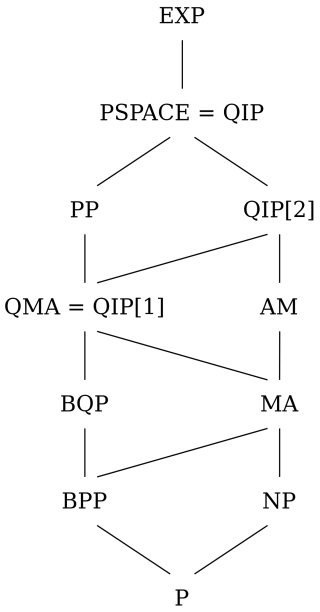

Figura 2. Inclusão de QIP[1], QIP[2] e QIP.

4. Problemas em aberto

A quantidade de relações em aberto é significativa, dada a dificuldade de se provar verdadeira, ou não, a equivalência ou pertinência entre classes de complexidade. Devido a isso serão apresentadas os principais pontos em aberto no campo da complexidade computacional, enfatizando relações em aberto mais conhecidas ou que tendem a serem provadas em um futuro próximo.

4.1. P vs NP

O problema se $P = NP$ é considerado por muitos o maior problema não resolvido na ciência da computação, segundo [Hemaspaandra 2002] a maioria dos cientistas da computação acredita que $P \neq NP$, e a razão para isso é que depois de décadas de estudo sobre estes problemas não foi possível achar um algoritmo polinomial que decida qualquer um dos mais de 3000 problemas $NP - Completos$ conhecidos.

4.2. P = BPP e NP = MA = AM

Ainda não sabemos se $P = BPP$, mas conjectura-se que sim, pois na maioria dos algoritmos aleatórios foi possível abrir mão da aleatoriedade e encontrar um algoritmo determinístico em *P*. Por razões similares conjectura-se que $NP = MA = AM$. Porém ainda não sabemos, por exemplo, se o problema do não isomorfismo em grafos, que está em *AM*, também está em *MA* ou *NP*.

4.3. Conjecturas sobre BQP

A relação entre *BQP* e *NP* é desconhecida e é um problema aberto até então, porém acredita-se que existem problemas em *BQP* que não estão em *NP* e problemas em *NP* que não estão em *BQP*.

Atualmente também é conjecturado que há problemas em *BQP*, por exemplo, o Algoritmo de Shor [Shor 1997], que não estão em *BPP*, e dessa forma $BPP \subsetneq BQP$.

Este trabalho apresentou um estudo sobre as principais classes de complexidade clássicas, probabilísticas e quânticas. Enfatizando além de suas definições formais e problemas contidos, as relações de equivalência entre as mesmas, provadas na literatura, criando um diagrama que possibilita uma visão geral do estado atual da hierarquia de classes de complexidade.

5. Considerações Finais

O presente trabalho apresentou a Máquina de Turing Quântica e seu funcionamento e resumiu as definições das classes de complexidade clássicas, probabilísticas e quânticas presentes na literatura e suas relações de contingência, apresentando-as visualmente através de um diagrama hierárquico de classes.

Entende-se que ao estudar e mostrar as relações entre classes de complexidade, pode-se compreender melhor os limites dos modelos computacionais sobre aspectos de tempo e espaço, tornando possível a comparações entre eles. E dessa forma, alinhamos as expectativas sobre a Computação Quântica.

Referências

Adleman, L. M., DeMarrais, J., and Huang, M.-D. A. (1997). Quantum computability. *SIAM Journal on Computing*, 26(5):1524–1540.

Babai, L. (1985). Trading group theory for randomness. In *Proceedings Of The Seventeenth Annual Acm Symposium On Theory Of Computing - Stoc '85*, pages 421–429. ACM Press.

Babai, L. and Moran, S. (1988). Arthur-merlin games: A randomized proof system, and a hierarchy of complexity class. *J. Comput. Syst. Sci.*, 36(2):254–276.

Bernstein, E. and Vazirani, U. (1997). Quantum complexity theory. *SIAM Journal on Computing*, 26(5):1411–1473.

Cobham, A. (1964). The intrinsic computational difficulty of functions. In *Proceedings Of The 1964 Congress On Logic, Mathematics And The Methodology Of Science*, pages 24–30.

Cook, S. (1971). The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158.

Deutsch, D. (1985). Quantum theory, the church-turing principle and the universal quantum computer. In *Proceedings Of The Royal Society A : Mathematical, Physical and Engineering Sciences*, volume 400, pages 97–117. The Royal Society.

Disco, C. and Van Der Meulen, B. (1998). *Getting new technologies together*. De Gruyter.

Edmonds, J. (1965). Paths, trees, and flowers. *Canadian Journal Of Mathematics*, 17:449–467.

Gill, J. T. (1977). Computational complexity of probabilistic turing machines. *SIAM Journal on Computing*, 4(6):675–695.

Goldwasser, S. and Sipser, M. (1986). Private coins versus public coins in interactive proof systems. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, STOC '86, page 59–68, New York, NY, USA. Association for Computing Machinery.

Hemaspaandra, L. A. (2002). Sigact news complexity theory column 36. *SIGACT News*, 33(2):34–47.

Jain, R., Ji, Z., Upadhyay, S., and Watrous, J. (2011). Qip = pspace. *J. ACM*, 58(6).

Kitaev, A. and Watrous, J. (2000). Parallelization, amplification, and exponential time simulation of quantum interactive proof systems. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, STOC '00, page 608–617, New York, NY, USA. Association for Computing Machinery.

Levin, L. A. (1973). Universal sequential search problems. *Probl. Peredachi Inf.*, 9:115–116.

Marriott, C. and Watrous, J. (2005). Quantum arthur-merlin games. *CoRR*, abs/cs/0506068.

Melo, B. L. M. d. and Christofoletti, T. V. D. (2003). Computação quântica : Estado da arte. pages 0–4.

Nielsen, M. A. and Chuang, I. (2010). *Quantum computation and quantum information*. Cambridge University Press.

Ozawa, M. (1998). On the halting problem for quantum turing machines.

Papadimitriou, C. H. (1994). *Computational complexity.* Addison-Wesley.

Pollachini, G. G. (2018). Computacão quântica: Uma abordagem para estudantes de graduação em ciências exatas.

Rabin, M. O. (1960). Degree of difficulty of computing a function, and a partial ordering of recursive sets. Technical Report 2, Hebrew University.

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

Shamir, A. (1990). Ip=pspace (interactive proof=polynomial space). In *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, pages 11–15 vol.1.

Shor, P. W. (1997). Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *Siam Journal On Computing*, 26(5):1484–1509.

Sipser, M. (2006). *Introduction on the Theory of Computation*. Course Technology Cengage Lerning, 2 edition.

Vereschchagin, N. K. (1992). On the power of pp. In *Proceedings of the Seventh Annual Structure in Complexity Theory Conference*, pages 138–143. IEEE.

Watrous, J. (1999). Pspace has 2-round quantum interactive proof systems.

Watrous, J. (2000). Succinct quantum proofs for properties of finite groups. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, FOCS '00, page 537, USA. IEEE Computer Society.

Watrous, J. (2003). Pspace has constant-round quantum interactive proof systems. *Theoretical Computer Science*, 292(3):575 – 588. Algorithms in Quantum Information Prcoessing.