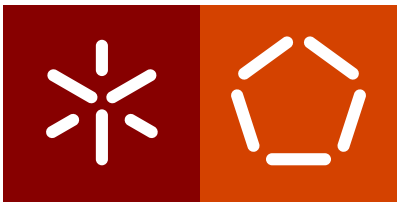
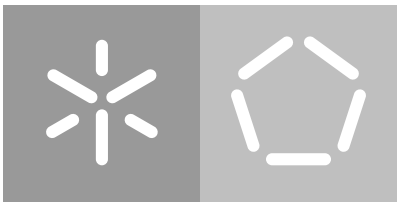


Universidade do Minho
Escola de Engenharia
Departamento de Informática

David Alves Campos Ferreira

Feynman Path-Sum Quantum Computer Simulator

April 2023



Universidade do Minho
Escola de Engenharia
Departamento de Informática

David Alves Campos Ferreira

Feynman Path-Sum Quantum Computer Simulator

Master dissertation
Master Degree in Physics Engineering

Dissertation supervised by
Luís Paulo Santos
Ernesto Galvão

April 2023

COPYRIGHT AND TERMS OF USE

ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisor prof. Luis Paulo Santos, and my co-supervisor Dr. Ernesto Galvão for all the support, patience and fruitful discussions throughout the development of this work.

I am also grateful to my family, especially my parents, for their unconditional support throughout all my life.

Finally, I would like to express my sincere gratitude to my girlfriend and my friends for their unwavering support and encouragement throughout the entirety of my Masters degree.

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration. I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

ABSTRACT

Classical quantum simulators are essential tools for studying quantum systems and simulating quantum algorithms. The hardware limitations of NISQ (Noisy Intermediate-Scale Quantum) devices make being able to build, test and run a quantum circuit/algorithm many times, and even test it under various noise scenarios, in a classical computer, extremely useful. Currently, the most prominent technique for classically simulating quantum circuits is known as Schrödinger type simulation. The memory usage of simulations using this technique increases exponentially with the number of qubits in a circuit, reaching prohibitive memory values relatively fast. This serves as motivation to investigate complementary classical quantum simulation techniques. The present work offers an investigation on how to improve the runtime of the Feynman path-sum approach for classical simulation of quantum circuits, taking into account the computational basis input and output states given and the branching structure generated by branching gates in a quantum circuit. The main contributions of this dissertation are two Feynman path-sum based simulation algorithms. These algorithms were able to successfully simulate quantum circuits with a large number of qubits (> 30) using polynomial space and it was demonstrated that the time complexity of these algorithms is more strongly influenced by the circuit structure rather than the circuit size.

Keywords: Quantum computing Quantum simulation Feynman

RESUMO

Os simuladores quânticos clássicos são ferramentas essenciais para o estudo de sistemas quânticos e simulação de algoritmos quânticos. As limitações de *hardware* dos dispositivos NISQ (*Noisy Intermediate-Scale Quantum*) tornam extremamente útil a possibilidade de construir, testar e executar um circuito/algoritmo quântico muitas vezes, e mesmo testá-lo sob vários cenários de ruído, num computador clássico. Actualmente, a técnica mais proeminente de simulação clássica de circuitos quânticos é conhecida como simulação do tipo Schrödinger. A utilização de memória das simulações que utilizam esta técnica aumenta exponencialmente com o número de qubits num circuito, atingindo valores de memória proibitivos com relativa rapidez. Este facto serve de motivação para investigar técnicas complementares de simulação quântica clássica. O presente trabalho oferece uma investigação sobre a forma de melhorar o tempo de execução do método de soma de caminhos de Feynman para a simulação clássica de circuitos quânticos, tendo em conta os estados, na base computacional, de entrada e saída e a estrutura de ramificação gerada pelas portas de ramificação num circuito quântico. As principais contribuições desta dissertação são dois algoritmos de simulação baseados na soma de caminhos de Feynman. Estes algoritmos foram capazes de simular com sucesso circuitos quânticos com um grande número de qubits (> 30) usando espaço polinomial e foi demonstrado que a complexidade temporal destes algoritmos é mais fortemente influenciada pela estrutura do circuito do que pelo tamanho do circuito.

Palavras chave: Computação quântica Simulação quântica Feynman

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1

INTRODUCTION

This dissertation focuses on the study and implementation of two simulator algorithms based on the Feynman path-sum method of classical simulation of quantum computers. Its structure and contributions are detailed in section 1.3 Before that, however, the whole area of work is put into context in sections 1.1 and 1.2. The first starts with a brief history of quantum computation, which aims at motivating a discussion on classical quantum simulation in section 1.2.

1.1 BRIEF HISTORY OF QUANTUM COMPUTING

The modern understanding of computer science was firstly proposed by Turing (1937), where he developed a mathematical model of computation, which is now called a Turing machine. These abstract machines are the mathematical groundwork of programmable computers, and Turing showed that there is a Universal Turing Machine that can be used to simulate any arbitrary Turing Machine. One of the many implications of this result is known as the Church-Turing thesis, which connects the concept of what classes of algorithms can be run in some physical device with the mathematical framework of a Universal Turing Machine. This article published by Turing instigated a sequence of events, which led to the rapid advancement of computer science.

However, the real turning point that marked the explosion of innovation in this field and the birth of modern electronics came with the invention of the transistor in 1947 by John Bardeen and Walter Brattain, paving the way for the digital age. This invention led to an unprecedented growth quantified by Moore (1965), known as Moore's law, stating that the number of transistors in a dense integrated circuit and ,therefore, its computational power doubles about every two years. Moore's law has roughly held true throughout the decades, due to the rapid scaling technology of the transistor. Nevertheless, in recent years, conventional fabrication methods are running into a problem of scale, as quantum effects begin to interfere more and more as the size of the devices becomes smaller.

In his 1959 lecture at the annual American Physical Society meeting at Caltech, titled "There's Plenty of Room at the Bottom: An Invitation to Enter a New Field of Physics",

Feynman (1959) recognized such a miniaturization was the way forward for computational hardware, and even predicted the problems quantum effects presented to computers with significantly small components. With an incredible stroke of insight, Feynman imagined that these effects could be exploited given the right computational paradigm.

Quantum computing began to assume its form in later work developed by Benioff (1980), where the earliest quantum mechanical model of a computer was described. In this article, Benioff showed that a computer working under the laws of quantum mechanics could be used to express a Schrodinger equation description of a Turing machine. Shortly after, Feynman (1982) hinted that the simulation of quantum systems was an exceedingly difficult task for a classical computer and proposed that, if it were possible to prepare a different, more controllable, quantum system, this could act as a specialized computer that could be used to determine the desired properties of the original system.

Three years later David Deutsch (1985) suggested that quantum computers could bring an advantage not only to the specific task of simulating quantum systems, but also to more general computational tasks, and proposed a quantum generalization of the Turing machine. Driven by the work of Turing, Deutsch questioned if a stronger version of the Church-Turing thesis could be derived from the laws of physics. The strong Church-Turing thesis states that any algorithmic process can be simulated efficiently using a probabilistic Turing machine, and Deutsch was set to define some device that could efficiently simulate an arbitrary physical system. Is still an open question whether Deutsch's formulation of a Universal Quantum Computer is sufficient for this purpose. However, what he accomplished, was a challenge to the strong Church-Turing thesis.

Following the work of the brilliant physicists aforementioned, excitement grew on what could be achieved with quantum computers and since the 1990's there have been extremely promising theoretical results. Even though of little practical use, one of the first examples of possible advantages a quantum computer may have over a classical one was presented by Deutsch and Jozsa (1992) when they developed a quantum algorithm, which determines if a function is constant or balanced, that is exponentially faster than its classical counterpart. Not much time after, Shor (1994) showed that the problem of finding prime factors of an integer and the discrete logarithm problem can be efficiently solved by a quantum computer. Most modern popular algorithms used for cryptography rely on the fact that the integer factorization or discrete logarithm problems are not solvable in time that grows polynomially with the size of the problem. Since this is no longer the case, a new field has emerged called post-quantum cryptography, whose purpose is to find suitable classical protocols for cryptography that cannot be efficiently broken by quantum computing. In parallel, Lov Grover (n.d.), would invent his Grover's algorithm, a quantum computer based algorithm able to speed up an unstructured database search quadratically. Due to their real world

applications, Shor's and Grover's algorithms brought a flood of intellectual curiosity to the area.

From then onwards, progress has been vast. From the first experimental demonstration of a quantum algorithm, realized in a working 2-qubit nuclear magnetic ressonance (NMR) quantum computer created by Chuang et al. (1998) and used to solve Deutsch's problem, to Google's claim (Arute et al. (2019)), to have achieved "Quantum Supremacy" (a term coined in 2012 by Preskill (2012) to express the goal of demonstrating that a programmable quantum device can solve a problem that no classical computer can solve in any feasible amount of time) with its 53-qubit Sycamore Processor. Currently quantum computation is living its NISQ (Noisy Intermediate-Scale Quantum) era, another term made up by Preskill (2018) to describe this pivotal new era in quantum technology. "Intermediate scale" indicates the size of quantum computers which will be available in the next few years, ranging from 50 to a few hundred qubits. "Noisy" emphasizes the imperfect control over these qubits, mainly due to their exquisitely sensitive nature. The technology to move beyond the NISQ era is unlikely to be available in the near future, but there are still many promising applications to delve deeper into using the NISQ technology, such as the development of hybrid algorithms that use NISQ devices, but make the best use of the limited quantum resources by implementing some parts of the algorithm in usual classical processors. Finally, another area of interest, and the subject of this dissertation, which is worth investigating is the classical simulation of quantum computers.

1.2 THE IMPORTANCE OF CLASSICAL QUANTUM SIMULATION

A classical quantum simulator is a software program which enables us to run quantum circuits on a classical computer as though they were running on a quantum computer. Nowadays, many of these simulators are available in the quantum ecosystem, as they have become increasingly popular tools in the world of quantum computing. Run-it-yourself simulators utilizing open-source tools like IBM's Software Development Kit *Qiskit* and Google's *Circ*, standalone, hardware-optimized packages such as *Intel-QS* and NVIDIA's *cuQuantum*, and cloud-based simulators from most major quantum cloud providers, including the 29 qubit cloud simulator provided by *The IonQ Quantum Cloud*, constitute examples of these.

Real quantum computer time, as well as the number of qubits available in today's quantum computers are scarce resources. Additionally, the noise in real quantum machines prevents them from running deep circuits. These limitations make being able to build, test and run a quantum circuit/algorithm many times, and even test it under various noise scenarios, without having to wait until the quantum computer is available after every change made in the circuit, extremely useful. Moreover, classical simulation of quantum computers helps shine more light on the incredibly puzzling computing power boundary between classical

and quantum computing. Simulators can also do things for developers that real quantum computers can't, such as look into a computation as it's occurring. In a real quantum system, it is not possible to see the complex state evolution that takes place during computation. All that can be done is initializing the qubits, perform gates and make a measurement. If we decide to try and peek at what's happening, not only do we destroy the in-progress computation by measuring it, we still don't see the "full" state, but just the measurement outcome. However, because a simulator is simulating this state classically, it's easy to pause it and look inside at any point to see the full quantum state, in a variety of different representations, as long as it's been programmed to do so.

The circumvention of all these hardware limitations doesn't mean that classical quantum simulators serve as any type of replacement of real quantum machines. A classical quantum simulator works by simulating the quantum operations of a computer, doing the same type of math, but at great computational cost. Where a quantum computer handles these operations natively, a classical computer must resort to perform complex and lengthy numerical calculations. In most simulators, every time a qubit is added, the state vector stored doubles its size. The growth is exponential and prohibitive memory complexity is reached very fast. This type of simulators are also known as Schrödinger-type simulators and currently, despite their inefficiency, they are the most prominent type of simulators. Consequently, classical computers performing Schrödinger-type simulations are only able to simulate quantum computers up to a point.

It is impossible for a simulator in any classical supercomputer in the world to match the performance of a real quantum computer with a sufficiently large number of qubits and, with the expected growth in the number of high quality qubits available in the industry's quantum computers, this difference will only become more palpable. Nevertheless, even then, classical quantum simulators will still be valuable isolated environments in which to test algorithms, debug code and learn how to think in quantum terms, and efforts must be made to further optimize and develop new simulation algorithms, so that they become even more powerful and useful. This serves as motivation to study other classical quantum simulation techniques.

This is where the Feynman path-sum method comes in as an alternative approach that allows for a more memory efficient simulation of large quantum systems. The Feynman path-sum method is based on the idea that the time evolution of a quantum system can be represented as a sum over all possible paths that the particles, or qubits, in the system can follow, and that each path is weighted according to its corresponding probability amplitude. Unlike a Schrödinger-type simulator, a Feynman path-sum simulator does not require memory that increases exponentially with the number of qubits.

1.3 OBJECTIVES, CONTRIBUTIONS AND STRUCTURE

The motivation behind the contents in this dissertation is to create an expanded overview on the topic of classical quantum simulation, by focusing on techniques to optimize the Feynman path-sum simulation approach. The techniques developed in this work are put into practice in the implementation of two different classical quantum simulators.

Chapter 2 gives a brief overview over some of the most used techniques for classical simulation of quantum computers. It begins with a discussion about weak and strong simulation, providing an introduction to the concepts, highlighting their differences and contextualizing Feynman path-sum based simulation algorithms in this framework of classical quantum simulation approaches. The simulation techniques included in this chapter are: Schödinger-type simulation, Feynman path-sum simulation, tensor network techniques, efficient classical simulation of restricted circuits, and ZX-calculus applied to the optimization of quantum circuits. For each of these techniques there is a brief introduction, with more focus on the Feynman path-sum approach, and important results in these fields are presented.

In chapter 3, an optimization method of the Feynman path-sum approach for classical simulation of quantum computers, that explores the structure of quantum circuits and the branching nature of quantum gates, is presented. Then, using this optimization method, the main original contributions of this thesis are presented and thoroughly discussed: two simulation algorithms, *PathNB* and *PathRec*. At last, the potential benefits of a pre-compilation step are discussed, and important results regarding this topic are presented.

Based on these two simulation algorithms, two quantum computer classical simulators were implemented, *SimulatorNB* and *SimulatorRec*. In chapter 4 the experimental setup and the results regarding the performance of these simulators are presented. This chapter provides results of correctness tests as well as a detailed performance comparison of the two simulators, with and without pre-compilation, for different types of quantum circuits, culminating in the presentation of results of the performance comparison of these two simulators when simulating circuits where a Schrödinger-type simulator would consume excessive amounts of memory.

2

TECHNIQUES FOR CLASSICAL SIMULATION OF QUANTUM COMPUTERS

The structure of this chapter is as follows. The first section presents a brief introduction to the concepts of weak and strong simulation, highlighting the advantages and disadvantages of each and contextualizing Feynman path-sum based simulation algorithms in this framework of classical quantum simulation approaches.

Sections 2.2 to 2.6 give a brief introduction to some of the most used and important techniques for classical simulation of quantum computers, except for section 2.3, where the Feynman path-sum approach is introduced with more detail. The techniques featured in these sections are the Schrödinger type simulation, Feynman path-sum simulation, tensor network techniques, efficient classical simulation of restricted circuits and ZX-calculus and its use in circuit optimization, respectively. For each of these, important research results are presented.

2.1 WEAK AND STRONG SIMULATION

The notion of classical simulation of quantum computers encompasses two fundamentally different types of simulation approaches with different levels of accuracy and efficiency in simulating quantum systems: weak and strong simulation.

- In weak simulation, the goal is to efficiently sample from the output probability distribution of a quantum computation using classical algorithms.
- In strong simulation, the goal is to compute the measurement probabilities or expectation values of a quantum computation using classical algorithms with high precision.

The main advantage of strong simulation is its ability to reproduce the entire quantum state of a system with high accuracy, including all complex quantum effects such as entanglement. This makes it a valuable tool for understanding the behavior of quantum systems and verifying the correctness of quantum algorithms. However, strong simulation is computationally demanding and requires exponentially more resources as the size of the system grows,

making it impractical for large-scale simulations. The Feynman path-sum based simulators implemented and discussed throughout this thesis fall into the strong simulation category.

In contrast, the main advantage of weak simulation is its computational efficiency, which allows for the simulation of large-scale quantum systems. Because quantum mechanics is inherently probabilistic, it is reasonable to argue that weak simulation is the more intuitive concept for classical simulation. Examples of quantum circuit classes for which strong simulation is intractable whereas weak simulation is achieved by elementary sampling methods are presented by den Nest (2009). However, weak simulation cannot reproduce the entire quantum state of a system with perfect accuracy and cannot capture the full range of quantum effects.

2.2 SCHRÖDINGER TYPE SIMULATION

In quantum computing, the basic unit of information is stored in a qubit, described in Dirac's bra-ket notation as $\alpha|0\rangle + \beta|1\rangle$ for complex numbers α, β satisfying $|\alpha|^2 + |\beta|^2 = 1$. The values α and β, and $|\alpha|^2$ and $|\beta|^2$ correspond to the probability amplitudes and probabilities of the qubit being in the state $|0\rangle$ and $|1\rangle$, respectively. Since every qubit has a probability of being either in state $|0\rangle$ or state $|1\rangle$, a system of n qubits has an associated amplitude for each of the 2^n possible bit string combinations of n bits.

When simulating a quantum algorithm running on n qubits, a Schrödinger type simulator, such as the Statevector Simulator from the *BasicAer* module of Qiskit, stores the quantum state of these qubits in a 2^n-size vector of complex valued amplitudes. This vector is updated successively, in place, by applying quantum transformations due to quantum gates for the case of quantum circuits, using linear algebra operations and other mathematical techniques. The following example illustrates how a naive and unoptimized implementation of a Schrödinger type simulator could carry out a simulation. Figure 1 shows a two-qubit circuit with two Hadamard gates around a CNOT gate. One could evaluate the effect of this

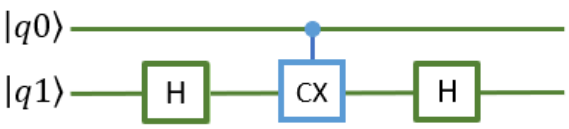


Figure 1: Quantum circuit diagram.

circuit on the input state vector by, first, ordering the gates left to right (parallel gates can be ordered arbitrarily), padding each gate with an identity matrix of an appropriate dimension

via Kronecker products to obtain a $2^2 \times 2^2$ matrix, and then multiply all those matrices in order.

$$(I \otimes H)CNOT(I \otimes H) \tag{1}$$

With $I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$ and $CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$.

The resulting operator, U

$$U = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \tag{2}$$

represents the entire circuit and can be multiplied by the input state vector to find the output state vector. While mathematically simple, this method is enormously wasteful and usually infeasible in practice. Instead, one applies each gate to the state vector, to avoid matrix-matrix multiplications. A key insight in high-performance Schrödinger type simulation is how not to pad gates with identity matrices (Fatima and Markov (2020)).

Applying each gate to the state vector requires iteration over all 2^n elements of the vector, therefore, for a n-qubit circuit consisting of l gates, the time and space complexities of this algorithm are $O(l2^n)$ and $O(2^n)$, respectively. The exponential growth in time and spatial complexity with the number of qubits makes this type of simulation classically inefficient. Nonetheless, Schrödinger type simulation is the mainstream technique for general-case simulation of quantum algorithms, circuits and physical devices, because

- it is commonly used for small and mid-size quantum-circuits and device/technology simulations because its unoptimized variants are relatively straightforward to implement;
- dominates supercomputer-based quantum circuit simulations because it can leverage distributed memory (De Raedt et al. (2019));
- has been extended for better scalability via layered simulation (Pednault et al. (2019)). For example, combining Schrödinger simulation with Feynman path-sum simulation enables scaling tradeoffs between circuit depth and width (Aaronson and Chen (2016)).

2.3 Feynman path-sum simulation

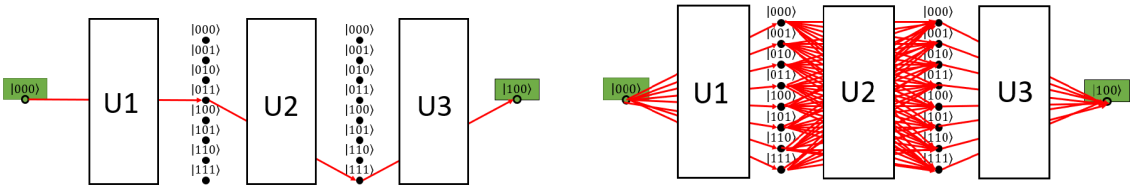
Feynman path integrals are a mathematical technique that can be used to classically simulate the behavior of quantum systems. This formulation of quantum dynamics was

developed by physicist Richard Feynman (1948) as a way to understand the quantum-mechanical behavior of particles and systems, and it has since become a widely used tool in quantum physics and quantum computing. The technique is based on the idea that the time evolution of a quantum system can be represented as a sum over all possible paths that the particles, or qubits, in the system can follow, and that each path is weighted according to its corresponding probability amplitude.

For a given circuit, C, with n qubits and d layers of unitary gates, the goal in a Feynman path-sum simulation is to find the probability amplitude of a certain state $|y\rangle$ being an output of C with a given input state $|x\rangle$, or:

$$\langle y| C |x\rangle = \langle y| U_d U_{d-1} ... U_2 U_1 |x\rangle . \tag{3}$$

A "path", ω, in a Feynman path-sum simulation of a quantum circuit connects the input state with the output state. As an example, figure 2a represents a random path in a 3-qubit circuit with 3 layers of unitary gates connecting the input state $|000\rangle$ to the output state $|100\rangle$, while figure 2b represents all possible paths for the same 3-layer circuit, and with the same fixed input, $|x\rangle$, and output, $|y\rangle$.



(a) A possible path.

(b) All possible paths.

Figure 2: Paths in a 3-qubit circuit with 3 layers of unitary gates connecting the initial state $|000\rangle$ to the final state $|100\rangle$.

Each path is associated with a probability amplitude, α_{ω_i}, that can be calculated as follows:

$$\alpha_{\omega_i} = \prod_{j=1}^{d} \left\langle Z_{i_{j+1}} \middle| U_j \middle| Z_{i_j} \right\rangle . \tag{4}$$

In this case, Z_{i_1} corresponds to the input state and $Z_{i_{d+1}}$ corresponds to the output state. Z_{i_2} to Z_{i_d} correspond to the intermediate states traversed by the path ω_i. U_1 to U_d correspond to the layers of unitary gates in the circuit. The probability amplitude, α_{ω_1}, corresponding to the configuration and path shown in figure 2a, but now with the specific quantum circuit shown in figure 3, can be calculated as follows:

$$\begin{aligned}\alpha_{\omega_1} &= \prod_{i=1}^{3} \left\langle Z_{1_{j+1}} \right| U_j \left| Z_{1_j} \right\rangle \\ &= (\langle 1| H |0\rangle \langle 1| X |0\rangle \langle 0| H |0\rangle)(\langle 1| S |1\rangle \langle 1| S |1\rangle \langle 1| X |0\rangle)(\langle 0| H |1\rangle \langle 0| X |1\rangle \langle 1| S |1\rangle) \\ &= \frac{i\sqrt{2}}{4}\end{aligned} \tag{5}$$

with $H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$, $X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ and $S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$.

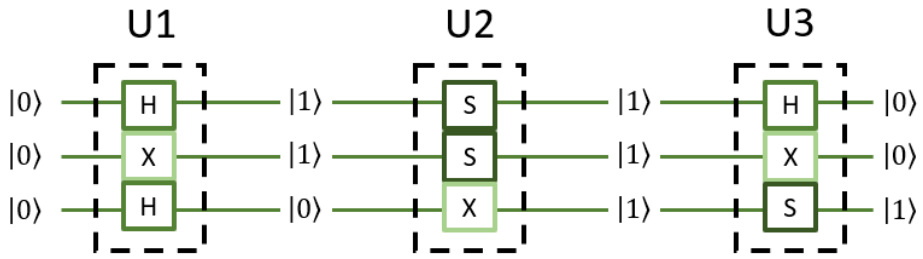


Figure 3: Quantum circuit with initial state $|000\rangle$, final state $|101\rangle$ and intermediate states corresponding to those in figure 1(a).

To find the probability amplitude $\langle y| U_d U_{d-1} ... U_2 U_1 |x\rangle$, the previous calculation must be done and summed across for all possible paths. For a circuit on n qubits, with d unitary layers, an initial state $|x\rangle$ and a final state $|y\rangle$, there are $2^{n(d-1)}$ possible paths, each with an associated probability amplitude, α_ω.

$$\langle y| U_d U_{d-1} ... U_2 U_1 |x\rangle = \sum_{i=1}^{2^{n(d-1)}} \prod_{j=1}^{d} \left\langle Z_{i_{j+1}} \right| U_j \left| Z_{i_j} \right\rangle . \tag{6}$$

For a quantum circuit with n qubits and m gates, eq.(6) shows that the Feynman path-sum simulation algorithm calculates an amplitude as a sum of terms. It does so using $\sim 4^m$ time and $\sim m + n$ space (Bernstein and Vazirani (1993)). Unlike the Schrödinger type simulation, the Feynman path-sum simulation does not require exponential memory with the number of qubits. In most cases $m \gg n$, and the difference between m and n can have significant practical implications. For instance, in Google's Sycamore quantum advantage experiment (Arute et al. (2019)), n had a value of 53, while m had a value of over 1500, which means that 2^n time is reasonable whereas 4^m time is not. This shows that, although the canonical version of the Feynman path-sum simulation algorithm, explained in this section, is more

memory-efficient than the Schrödinger type simulation algorithm, due to the exponential growth of paths with the number of gates, it becomes exponentially more computationally expensive with increasing circuit depth.

In fact, Google's Sycamore experiment used a hybrid Schrödinger–Feynman (Markov et al. (2018)) algorithm for circuits with more than 43 qubits that breaks the circuit up into two sub-circuits of qubits and efficiently simulates each sub-circuit using a Schrödinger method, performing a full wave-function evolution of the sub-circuits for all the paths defined by the partition, before connecting them using an approach similar to that of the Feynman path-integral. Another example of the use of the Feynman path-sum approach for simulating quantum circuits is described by Boixo et al. (2017), where circuits are described using the Feynman path-sum notation, similar to eq.(6), then mapped to an undirected graphical model with complex factors and simulated using a variable elimination algorithm. Huang and Love (2021), presented an algorithm that also uses the Feynman path-sum approach, but using stabilizer projector decomposition of unitaries. More specifically, this algorithm uses a recursive form of the Feynman path-sum approach, detailed below.

Finally, an algorithm that will be relevant for later discussion is the Feynman path-sum recursive algorithm proposed by Aaronson and Chen (2016). To reduce the time cost of the Feynman path-sum approach, one can notice that there is a large number of repeated calculations in eq.(6), if the terms in the sum are calculated path by path. For example, if two of the paths are the same for the first $d-1$ steps of the product, but only differ at the last step. In eq.(6), these two paths will give two terms that only differ on one factor of the product. However, naive evaluation of eq.(6) results in redundant computation of the $d-1$ factors. One can avoid these repeated calculation as follows. By first slicing a circuit into two sub-circuits, C1 and C2:

$$\langle y| C |x\rangle = \sum_{z\in(0,1)^n} \langle y| C_2 |z\rangle \cdot \langle z| C_1 |x\rangle \tag{7}$$

one can obtain a recursion relation for the time cost of calculating the whole sum from the results of the two sub-circuits:

$$T(d) = 2^{n+1}T(d/2) \tag{8}$$

assuming the depth of the whole circuit is d and the depth of both C1 and C2 is $d/2$. Following this relation, one can recursively divide the two sub-circuits further until getting down to single-layer circuits. In this way, the sum can be calculated more efficiently than it would be with the canonical Feynman approach, without sacrificing much of the space cost advantages of the Feynman approach. In fact, there are only $O(log_2(d))$ steps to reach the last level of recursion, where the sub-circuits haven depth $d = 1$ (single layer), and at each step a n-bit string y is needed to label the term that is being computed. Therefore, this algorithm needs $O(nlog_2(d))$ space to recursively return a single term to the whole

summation. Meanwhile, one can see the total time cost is brought down to $O(n2^{nlog_2(d)})$ by solving the recursion relation in eq.(8). The idea of this algorithm will be used later, in section 3.3, in the development of a different version of a recursive Feynman path-sum algorithm.

2.4 TENSOR NETWORK TECHNIQUES

A common approach for applying gates is to do so in the form of matrix multiplications. For example, the effect of applying a unitary operator, A, followed by a unitary operator, B, on a given state, ψ, can be done in a naive way using $B \times A$ and then multiplying the result with the state vector, $|\psi\rangle$. In a state with n qubits, the amplitudes may be represented by a 2^n-sized state vector, $|\psi\rangle$, and A and B by two $2^n \times 2^n$ matrices. The best known asymptotic complexity of multiplying two $2^n \times 2^n$ matrices is $O(n^{2.37188})$ (Duan et al. (2022)). An improved way to perform this operation is to start by multiplying the $2^n \times 2^n$ unitary operator A with the 2^n-sized state vector $|\psi\rangle$ and then, multiply the $2^n \times 2^n$ unitary operator B with the 2^n-sized modified state vector. Although the time complexity decreases by performing the operations in this order, it still reaches a prohibitive number of instructions relatively fast, with the increase in the number of qubits. For a 20 qubit state, the state vector has $2^{20} = 1048576$ elements. Applying a unitary operator to this state is done by multiplying a $2^{20} \times 2^{20}$ matrix with the state vector with 2^{20} elements.

The exponential growth with the number of qubits detailed above served as motivation to develop new, more efficient representations of quantum states and quantum circuits. One of such representations are tensor networks. A tensor is a multi-dimensional array of numerical values. Tensors can be of any rank, with rank-0 being a scalar, rank-1 being a vector, and rank-2 being a matrix. Higher-rank tensors are simply multi-dimensional arrays of numbers, with the dimensionality determined by the rank. Tensor networks are a mathematical framework for representing and manipulating large quantum systems using tensors. They are based on the idea of representing quantum states and quantum operations as tensors. These tensors are then manipulated and contracted to simulate the application of quantum gates to quantum states, allowing for the efficient simulation of large quantum systems. A given quantum circuit can always be represented as a tensor network, where a quantum state of n qubits can be represented using a rank-n tensor, where each index of the tensor corresponds to a qubit. The elements of the tensor are the amplitudes of the state in the computational basis. One-qubit gates are rank-2 tensors (tensors of 2 indices with dimension 2 each), two-qubit gates are rank-4 tensors (tensors of 4 indices with dimension 2 each), and in general n-qubit gates are rank-$2n$ tensors. The computational and memory cost for the contraction of such networks is exponential with the number of open indices and, for large enough circuits, the network contraction is unpractical; nonetheless, it is always

possible to specify input and output configurations in the computational basis through rank-*1* Kronecker deltas over all qubits, which can vastly simplify the complexity of the tensor network. This representation of quantum circuits gives rise to a high performance simulation technique, first introduced by Markov and Shi (2008), where the contraction of the network gives amplitudes of the circuit at specified input and output configurations.

An essential problem in tensor network contraction is to determine an order for contracting tensors pair by pair, which is referred to as contraction order. The tensor networks associated with quantum circuits are usually irregular, so it is a difficult problem to find an optimal order for contracting the whole network. Markov and Shi (2008) showed that one can use the optimal tree decomposition of the line graph associated with the tensor network to find the optimal contraction order. However, finding the optimal tree decomposition for a general graph is a NP-hard problem. Pan and Zhang (2021) developed a tensor network method for simulating quantum circuits, which they called Big Head simulation, that uses an order-finding algorithm with reduced computational cost and using this method the authors managed to simulate the Google Quantum AI experiment on quantum computational advantage, using the Sycamore chip.

There are several types of tensor networks, each with their own unique properties and applications. Some of the most well-known types include:

- Matrix Product States (MPS): A method for representing a quantum state as a network of tensors, where each tensor represents a quantum state of a subsystem, and the indices of the tensors are connected to represent the entanglement between the subsystems. The MPS formalism is described in detail by Vidal (2003). An example of a simulation using MPS is given by Dang et al. (2019) where they optimised high-level classical simulations of Shor's quantum factoring algorithm and performed a matrix product state simulation of a *60*-qubit instance of Shor's algorithm that would otherwise be infeasible to complete without an optimised entanglement mapping.

- Projected Entangled Pair States (PEPS): A method for representing a quantum state as a two-dimensional network of tensors, where each tensor represents a quantum state of a region in a two-dimensional lattice, and the indices of the tensors are connected to represent the entanglement between the regions. Guo et al. (2019) developed a general purpose quantum simulator that uses PEPS.

- Multi-scale Entanglement Renormalization Ansatz (MERA): A method for representing a quantum state as a network of tensors, where each tensor represents a quantum state of a subsystem, and the indices of the tensors are connected to represent the entanglement between the subsystems in different scales. A protocol for MERA-based classical simulation of arbitrary quantum circuits is presented by Luchnikov et al. (2021). They demonstrated that their approach can be used for successful simulation

of intermediate-size quantum circuits $n = 27$ qubits. They also discussed estimations showing that the MERA tends to outperform MPS-based simulators for a large number o qubits.

2.5 EFFICIENT CLASSICAL SIMULATION OF RESTRICTED CIRCUITS

Efficient classical simulation of restricted circuits is a topic of research in quantum computing that aims to find efficient algorithms for simulating the behavior of certain restricted classes of quantum circuits on a classical computer. One such type of restricted circuits are the stabilizer circuits.

Theorem 1 (Gottesman-Knill Gottesman (1998)) *A quantum circuit using only the following elements can be simulated efficiently, that is, in time that scales polynomially with the circuit size, on a classical computer:*

1. *Preparation of qubits in computational basis states,*
2. *Clifford gates (Hadamard gates, CNOT gates, phase gate S), and*
3. *Measurements in the computational basis.*

Such circuits can generate huge amounts of entanglement, and can be used for superdense coding, quantum teleportation, quantum error-correcting codes, etc. Stabilizer circuits are circuits corresponding to the description given above, in the statement of the Gottesman-Knill theorem. This group of gates is called the Clifford group. Aaronson and Gottesman (2004) showed that these type of circuits can be simulated efficiently in polynomial time ($O(n^2)$). Aaronson and Gottesman also leverage their classical simulation algorithm for Clifford circuits to simulate general quantum circuits, with a slow-down that is exponential in the number of non-Clifford gates. Based on this idea, Bravyi et al. (2019), developed a simulation algorithm that can decompose a quantum circuit constituted by Clifford gates and arbitrary diagonal gates in stabilizer circuits, with the computational cost scaling with the number of non-Clifford gates. This latter approach is the starting point of other recent developments, known as stabilizer-rank methods for classical simulation of general circuits.

Another type of circuits with efficient classical simulation are circuits with limited entanglement. For example, pure states of n qubits, which are always unentangled, can easily be simulated classically, since one only needs to write $2n$ amplitudes, rather than 2^n, on each time step. Considering a pure state $|\psi\rangle \in H_2^{\otimes n}$ of an n-qubit system, where A denotes a subset of the n qubits and B the rest of them, entanglement in a such a state can be measured in the following way:

$$|\psi\rangle = \sum_{\alpha=1}^{\chi_A} \lambda_\alpha \left|\Phi_\alpha^{[A]}\right\rangle \otimes \left|\Phi_\alpha^{[B]}\right\rangle \tag{9}$$

where the vector $\left|\Phi_\alpha^{[A]}\right\rangle$ ($\left|\Phi_\alpha^{[B]}\right\rangle$) is an eigenvector with eigenvalue $|\lambda_\alpha|^2 > 0$ of the reduced density matrix $\rho^{[A]}(\rho^{[B]})$, whereas the coefficient λ_α follows from the relation $\left\langle\Phi_\alpha^{[A]}\middle|\psi\right\rangle = \lambda_\alpha \left|\Phi_\alpha^{[B]}\right\rangle$. The Schmidt rank χ_A is a natural measure of the entanglement between the qubits in A and those in B (Vidal (2003)). Accordingly, the entanglement of state $|\psi\rangle$ is quantified by χ,

$$\chi = max_A \chi_A \tag{10}$$

that is, by the maximal Schmidt rank over all possible bipartite splittings A:B of the n qubits. It turns out that the Schmidt rank can be computed efficiently by diagonalizing the density matrix on one of the two sides, and finding how many different eigenvalues it has (this will not be efficient in the number of qubits, as the reduced density matrix is exponential-size with relation to this number). In general, the Schmidt rank may be as high as $2^{n/2}$. In some cases, when it is small (polynomially bounded) one can efficiently simulate the circuit. This is the main theorem of Vidal's paper.

Theorem 2 (Vidal (2003)) *Suppose a quantum computation involves nearest-neighbor interactions only among qubits on a line, and that χ_{max} is polynomially bounded at every step, then the computation can be efficiently simulated on a classical computer.*

The simulation is essentially carried out via dynamic programming – for each qubit, a $\chi \times \chi$ matrix encoding how each qubit interacts with the other qubits is stored (using contraction of tensors to obtain a more compact representation). This compact representation can be locally updated for nearest-neighbor operations, and the probabilities can be obtained from them. Thus, one only needs to store $O(n\chi^2)$ amplitudes on each time step to simulate these circuits. This algorithm is not meant for simulating physical systems, but rather quantum circuits, i.e., applying nearest-neighbor unitaries. Vidal later developed a related algorithm for simulating physical systems with nearest-neighbor Hamiltonians, or for estimating the ground states where there is limited entanglement (Corboz et al. (2010)), and then was able to use this algorithm to solve some problems in condensed-matter physics where the states of interest had dimensions too high to deal with using existing techniques. It turns out that cases with low entanglement encompass most cases that physicists care about, since creating entanglement is hard— this is, after all, what makes building quantum computers so hard in the first place.

A class of restricted circuits introduced by Valiant (2002) is another example of circuits with efficient classical simulation. The class includes a special set of unitary 2-qubit gates on nearest-neighbor qubits, called matchgates. Terhal and DiVincenzo (2002) presented a slightly restricted version of the main theorem of Valiant's paper:

Theorem 3 (Valiant (2002)) *Let M be the unitary transformation representing a quantum circuit on n qubits that consists of 2-qubit gates U on qubits x_i and x_{i+1}, $i = 0, ..., n-1$, where $e^{i\phi}U$ is of the form*

$$e^{i\phi}U = \begin{bmatrix} U_{11}^1 & 0 & 0 & U_{12}^1 \\ 0 & U_{11}^2 & U_{12}^2 & 0 \\ 0 & U_{21}^2 & U_{22}^2 & 0 \\ U_{21}^1 & 0 & 0 & U_{22}^1 \end{bmatrix}$$

where U^1 and U^2 are arbitrary elements of $SU(2)$ and ϕ is an arbitrary phase. There exist polynomial-time classical algorithms that evaluate (1) $|\langle y| M |x\rangle|^2$ for arbitrary bitstrings x and y, (2) $Tr\langle y^*| M |x\rangle \langle x| M^\dagger |y^*\rangle$, where y^* corresponds to an assignment of an arbitrary k-bit subset for any k, and (3) sample, given an arbitrary input string $|x\rangle$, the probability distribution over outcomes y^* of a measurement (in the computational basis) on an arbitrary k-bit subset of the qubits.

Terhal and DiVincenzo (2002) showed that this class of efficiently simulatable quantum computations corresponds to a physical model of non-interacting fermions in one dimension. Another interesting result in regard to circuits constituted by matchgates was presented by Jozsa et al. (2009). In this paper, Jozsa and his collaborators showed an equivalence between matchgate circuits on n qubits and general quantum circuits on $O(log(n))$, by presenting the following theorem and its proof.

Theorem 4 (Jozsa et al. (2009)) *The following equivalence between matchgate circuits and general quantum circuits holds:*

- *Given a matchgate circuit, MG, of nearest-neighbor matchgates with an n-qubit input $|x_1...x_n\rangle$, m gates, and final measurement on qubit k, there exists an equivalent quantum circuit, C, with an input of $[log(n)] + 3$ qubits initialized in the 0 state, composed of $O(mlog(n))$ gates, and with final measurement on the first qubit. Moreover, the encoding of the circuit C can be computed from the encoding of the matchgate circuit MG by means of a classical space $O(log(n))$ computation.*
- *Conversely, given any quantum circuit, C with an n-qubit input $|y1...yn\rangle$, m gates, and final measurement on the first qubit, there exists an equivalent matchgate circuit, MG, with an input of $2^n - 1$ qubits initialized in the 0 state, composed of $O(m2^{2n})$ gates and with final measurement on the first qubit. Moreover, the encoding of matchgate circuit MG can be computed from the encoding of the circuit C by means of a classical space $O(n)$ computation.*

The interested reader is referred to Brod (2014) for an in depth review on matchgates and matchgate circuits, and more recent results related to this model of quantum computation.

2.6 ZX-calculus

The ZX-calculus is a graphical language and formalism for quantum computing and quantum information, which was introduced by Coecke and Duncan (2008). It provides a way to reason about linear maps between qubits using diagrams called ZX-diagrams. These diagrams consist of nodes, which are either green or red and represent quantum states connected by wires that represent quantum operations. These nodes are called *spiders*. The wires can curve and cross each other, and multiple wires can connect to the same node. Wires entering the diagram from the left are inputs, while wires exiting to the right are outputs. In addition, there are also special nodes called Hadamard nodes, which are represented by yellow boxes and always connect to exactly two wires.

ZX-diagrams can represent linear maps between qubits in a graphical format in a similar way as quantum circuits represent unitary maps between qubits, but these two models differ in two main ways. Firstly, ZX-diagrams are not bound to the rigid topological structure of circuits and can be deformed in any way. Secondly, ZX-diagrams come with a set of rewrite rules, called the ZX-calculus, which enable calculations to be performed directly on the graphical language itself. This means that the ZX-calculus allows for calculations to be performed using simple, visual transformations of the diagram. This can make complex calculations easier to perform and can lead to insights that might not be immediately obvious in traditional mathematical notation. For a complete review on ZX-calculus and its rules the interested reader is referred to van de Wetering (2020). The core rules of the ZX-calculus give a sound and complete theory for Clifford circuits (Backens (2014)) and, more interestingly, Jeandel et al. (2018) have shown that the ZX-calculus expanded with four new axioms becomes complete for the universal family of circuits Clifford + T.

In recent years, there has been a surge of interest in the field of circuit optimization utilizing the ZX-calculus. Duncan et al. (2020) present an approach to quantum circuit optimization, based on the ZX-calculus, in which the authors first interpret quantum circuits as ZX-diagrams and then, using the rules of the ZX-calculus, they give a simplification strategy for ZX-diagrams based on the two graph transformations of local complementation and pivoting and show that the resulting reduced diagram can be transformed back into a quantum circuit. The underlying graph of their simplified ZX-diagram always has a graph-theoretic property called generalised flow, which in turn yields a classical circuit extraction procedure. Kissinger and van de Wetering (2022) combined this optimization method with the sum-of-stabilisers method to develop an enhanced technique for strong classical simulation of quantum circuits. With this technique the authors were able to successfully simulate random 50 and 100-qubit Pauli exponential circuits with up to 70 T gates. as well as 50-qubit hidden shift circuits with up to 1400 T gates.

The ZX-calculus is a powerful tool for reasoning about quantum circuits and has demonstrated significant potential for circuit optimization. By representing quantum circuits as ZX-diagrams and applying the rewrite rules of the ZX-calculus, researchers can manipulate circuits in a graphical and intuitive manner, leading to the discovery of new optimization techniques and insights into the properties of quantum circuits. However, while the ZX-calculus has shown promising results for certain types of circuits, there is still much work to be done in order to fully leverage its potential for optimizing complex quantum circuits. Further research is needed to explore the limits of the ZX-calculus and to identify new applications for this powerful tool in the field of quantum computing.

3

CLASSICAL SIMULATION BASED ON FEYNMAN PATH SUMS

In this chapter, the first section presents an optimization method of the Feynman path-sum approach for classical simulation of quantum computers, that explores the structure of quantum circuits and the branching nature of quantum gates. Throughout section 3.2 the development of a simulation algorithm, *PathNB*, based on the optimization method discussed in the previous section, is presented and exemplified. In section 3.3 a recursive simulation algorithm, *PathRec*, also based on the optimization method discussed in section 3.1, is explained and exemplified. In the last section there is a discussion about the potential benefits of a pre-compilation step. Important research results regarding this topic are presented and discussed.

3.1 OPTIMIZATION METHOD

In section 2.3 the canonical version of the Feynman path sum algorithm for simulation of a general quantum computer was described. In this section we will discuss ways to optimize this algorithm, having in mind circuits with certain specific characteristics. The first optimization of the canonical version of the Feynman path-sum simulation algorithm was to note that, for a given circuit, there can be many paths that contribute with a null amplitude and therefore, the goal is to compute only the path amplitudes that effectively contribute to the path sum, and not lose computational time calculating amplitudes with null contribution.

Considering again the example presented in section 2.3, it is straightforward to identify paths that contribute with null probability amplitude- the key point here is to look at the

specific form of single- and two-qubit gates used. Figure 4 shows some of these. Because of the effect of the X gate:

$$
\begin{aligned}
X &= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \\
X|0\rangle &= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle \\
X|1\rangle &= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle \\
\langle 0|X|0\rangle &= \langle 0|1\rangle = 0 \\
\langle 0|X|1\rangle &= \langle 0|0\rangle = 1 \\
\langle 1|X|0\rangle &= \langle 1|1\rangle = 1 \\
\langle 1|X|1\rangle &= \langle 1|0\rangle = 0
\end{aligned}
\tag{11}
$$

every path connecting the initial state of the second qubit, $|0\rangle$, with an intermediate state of the second qubit in state $|0\rangle$ is going to contribute with a null probability amplitude, as it would if both the initial and intermediate state were in state $|1\rangle$.

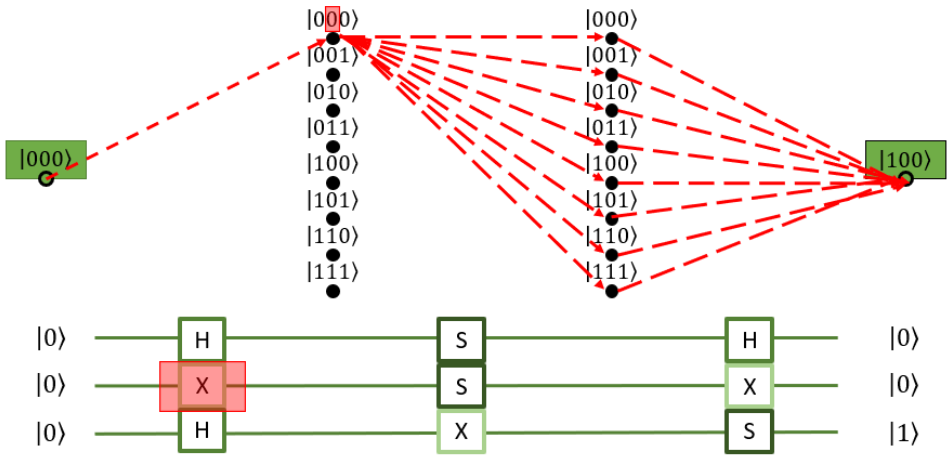


Figure 4: Example of paths with null amplitude contribution in a 3-qubit circuit with 3 layers of unitary gates connecting the initial state $|000\rangle$ to the final state $|101\rangle$.

The X gate maps a single classical state to another single classical state. The output state of an X gate acting on a classical input state is, therefore, classical. This effect is a result of the gate matrix having only one non-zero entry per row, and all the gates with this matrix property can be defined as non-branching. Examples of these include the identity, Pauli gates, CNOT and Toffoli gates, meaning that each of these gates map a single classical state to another single classical state (while possibly introducing a phase factor). On the other hand, when the gate matrix has more than one non-zero entry per row, then the gate is

capable of mapping a given state to a superposition of states, and all the gates with this matrix property can be defined as branching. A simple and common example of a branching gate is the Hadamard gate:

$$
\begin{aligned}
H &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \\
H|0\rangle &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \\
H|1\rangle &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)
\end{aligned}
\tag{12}
$$

As shown in eq.(12), the Hadamard gate takes each of the $|0\rangle$ and $|1\rangle$ states into an equal superposition of both states, and therefore "branches" a given computational basis state into a superposition of more than a single computational basis state.

For the following discussion only the Clifford + T gate set will be considered. In this gate set the only branching gate is the H gate. The notion of branching and non-branching gates is intimately related to the notion of contributing paths in a given circuit. Based on this interpretation of quantum gates, a visualization of the possible contributing paths can be achieved using a circuit graph representation. The input state $|x\rangle$ is drawn as a single node. Traversing the layers of unitary gates in the circuit, in order, from input to output, the graph is drawn taking into account the effect of each gate of the layer on their corresponding qubits, but without considering phase factors. If a layer is comprised only of non-branching gates, a single edge is drawn representing the single mapping of the gates on the previous state. However, if the layer has a number n of one-qubit branching gates, then 2^n edges are drawn corresponding to the number of states the previous state was mapped to as a result of the action of these gates. The counting 2^n assumes that the branching is always two-fold, which is true for the H gate and, consequently, for the gate set we are considering, but won't be true in general. At the end of each edge, another node is drawn and labeled with the single classical state produced as a result of traveling down that edge. After traversing all the layers of gates, the single classical states labeling the nodes drawn after the last layer represent some of the possible output states, $|y_0\rangle$ to $|y_m\rangle$, for a given input state. It can be possible to have different nodes with the same classical state label, after the last layer of gates. The number of paths in the graph connecting the input state $|x\rangle$ node to nodes with the same classical state label $|y_i\rangle$ after the last layer of gates, represents the number of paths connecting the input state, $|x\rangle$, to the output state, $|y_i\rangle$. An important remark is that not all the nodes drawn after the last layer of gates correspond necessarily to output states that yield $\langle y_i| C |x\rangle \neq 0$, because some pairs of paths have symmetric probability amplitudes and thus cancel out. As an example, considering the circuit shown in figure 4, a graph

representing the circuit and highlighting the possible contributing paths from its input state, $|000\rangle$, to its output state, $|100\rangle$, is drawn in figure 5.

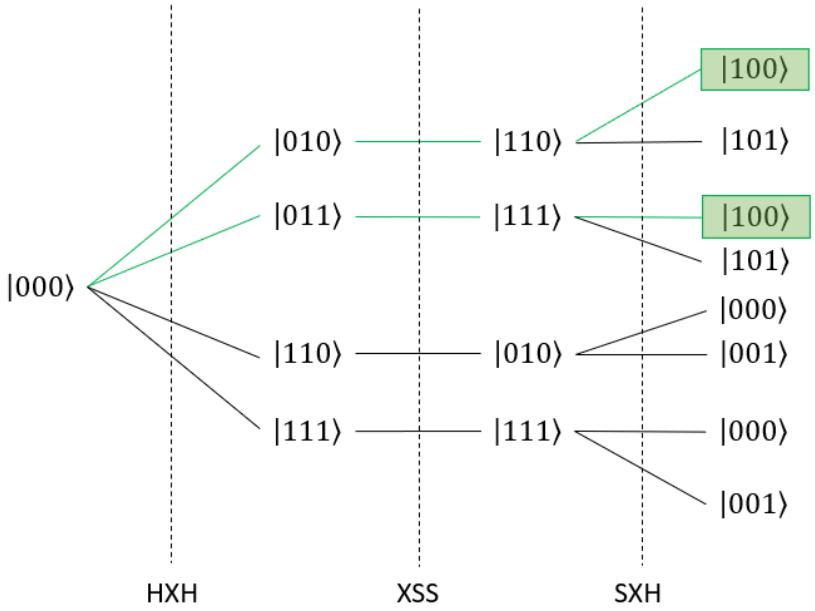


Figure 5: Graph representing the circuit shown in figure 3 and highlighting the contributing paths from its input state, $|000\rangle$, to its output state, $|100\rangle$.

In the analysis made in section 2.3, it was noted that a general circuit on n qubits and with m layers of unitary gates has $2^{n(m-1)}$ possible paths connecting its input state to its output state. For the circuit shown in figure 4, n=3 and m=3, which gives a total of 64 possible paths. The graph representation shown in figure 5 indicates that a maximum of only 2 of the possible 64 paths effectively contribute with a non-null probability amplitude. This means that a simulation of this circuit using the canonical version of the Feynman path-sum algorithm would spend approximately 97% of its time calculating probability amplitudes that do not contribute to the sum show in eq.(6).

Taking this into consideration, an algorithmic method that approximates the number of paths whose amplitude is calculated, to the number of effectively contributing paths, was developed. This method makes use of the circuit structure and explores the notion of propagation of certainty, or *classicality*, throughout the circuit. The concept of "propagation of classicality" refers to the process by which a quantum operation acts on a classical state and generates another classical state as the output. Given an arbitrary circuit, constituted of both branching and non-branching gates, the first step is to distinguish intermediate states, i.e. qubit states in the computational basis between unitary layers of gates, that are fixed at the outset, and intermediate states that are not. To achieve this, the circuit is swept forwards, from the input to the output, layer after layer, identifying intermediate states that

are fixed because of i) fixed global circuit input configuration, and ii) non-branching gate action. These intermediate states are colored green. The remaining intermediate states are colored red. The circuit's input state can be thought of as a green-colored state, since it is fixed. Figure 6 shows the rules for the coloring sweep, for one and two-qubit gates.

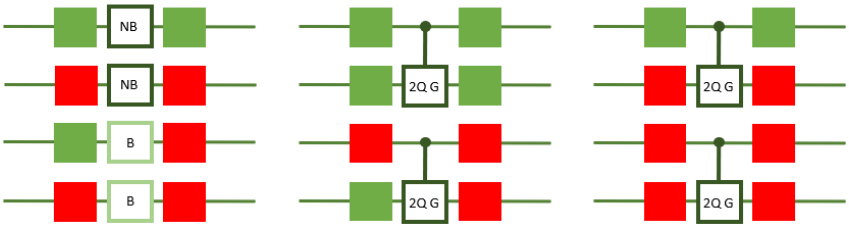


Figure 6: Rules for the coloring sweep for non-branching gates (NB), branching gates (B), and 2-qubit gates controlled$-U$, where U is a non-branching gate (2Q G).

The output state of a non-branching gate acting on a single qubit with a classical input state (green) is also classical, so it is colored green. Since the output state of a non-branching gate acting on a single qubit depends only on its input state, if its input state is not fixed, or classical, (red), its output state is not classical, and so it is colored red. The superposition states that result from applying a branching gate to a single qubit are not classical by definition, therefore, the output state of a branching gate acting on a single qubit is always not classical, so it is colored red regardless of its input state. The two-qubit gates controlled$-U$, where U is a non-branching gate, such as the CNOT, CZ or CY are non-branching, so if both their input states are fixed, so are their output states, resulting in a green coloration. If the input state of the control qubit is not classical, the output states of the control and target qubits are also not classical, since both these states depend on the input state of the control qubit, so they are colored red. If the input state of the control qubit is classical, so is its output state and, therefore, it is colored green. If the input state of the target qubit is not classical, neither is its output state and, therefore, it is colored red. Finally, if the input states of both the control and target qubits are not classical, neither are their outputs, and so they are colored red. Figure 7 presents the evolution of sweeping forwards the circuit represented in figure 4, from input to output.

The goal in a Feynman path-sum simulation is to find the probability amplitude presented in eq.(3), which shows that the output state is fixed and can also be thought of as a green-colored state. This means that it is also possible to propagate *classicality* from output to input, sweeping the circuit backwards and following the coloring rules shown in figure 6. Figure 8 presents the evolution of sweeping backwards the circuit represented in figure 3.

Assuming the user does not know if the chosen output state is indeed an output state which gives a non-zero value to eq.(6), it is not computationally efficient to carry out

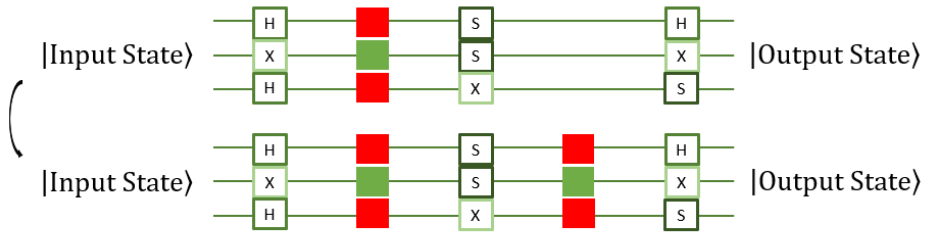


Figure 7: Evolution of a forward sweep of the circuit, using colors to identify the propagation of *classicality*.

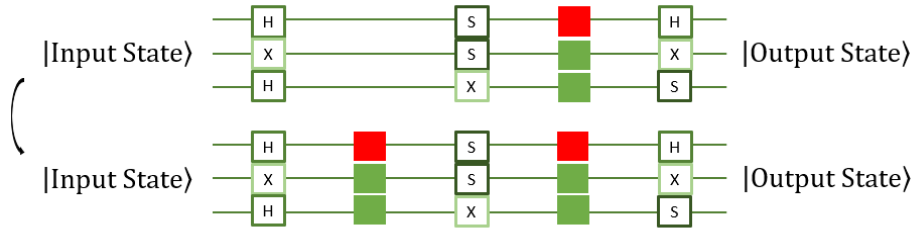


Figure 8: Evolution of a backwards sweep of the circuit, using colors to identify the propagation of *classicality*.

the whole algorithm with the possibility of ending up with a null probability amplitude. Considering this, a routine that searches for inconsistencies between the input and output states propagation was implemented and should be used at this point. By inspecting the graph represented in figure 6, one can note that, if the output state is different from any of the states at the nodes after the last layer, then eq.(6) will surely be zero. In the following example, it is assumed the user has chosen the output state $|111\rangle$. The first step of this routine is to calculate the fixed intermediate states that were a result of sweeping the circuit forward, starting from the input state and mapping successively each qubit state according to the action of the gates whose output state is colored green. The non-classical states, i.e. states in superposition, are represented as $|nd\rangle$. In this example, the result of carrying out this step is shown in figure 9.

Then, this step is repeated, but this time calculating the fixed intermediate states that were a result of sweeping the circuit backwards, starting from the output state and mapping successively each qubit state according to the action of the gates whose, in this case, input state is colored green. The not classical states are not defined, $|nd\rangle$, as well. In this example, the result of carrying out this step is shown in figure 10.

Finally, a check for inconsistencies between the two sets of intermediate states calculated is carried out. This is done by comparing the intermediate states of each set, in order. If a certain intermediate state has two different values, this represents an inconsistency and

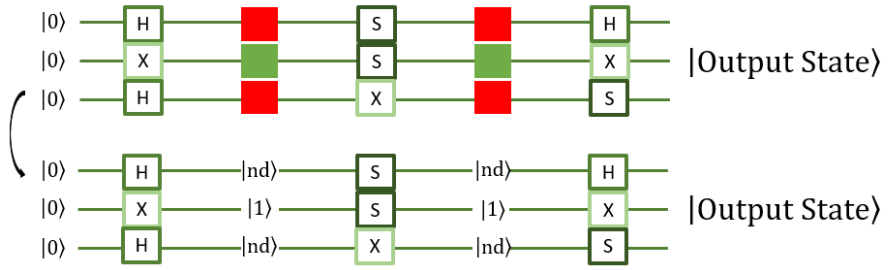


Figure 9: Calculation of the classical intermediate states after the forward sweep of the circuit, starting from the input.

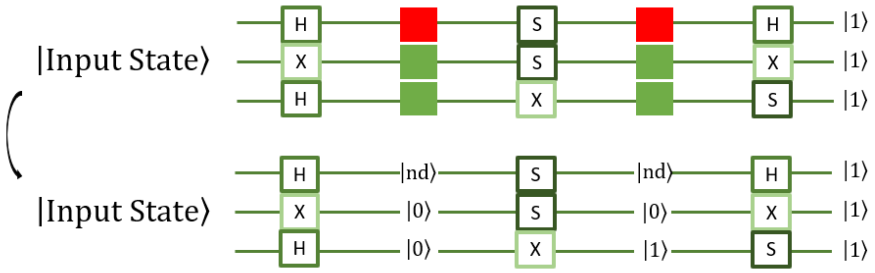


Figure 10: Calculation of the classical intermediate states after the backwards sweep of the circuit, starting from the output.

it implies that the output state chosen makes the amplitude eq.(6) zero. The non-classical states, $|nd\rangle$, are compatible with both $|0\rangle$ and $|1\rangle$. In this example it is immediate to see that the intermediate states of the second qubit have different values in each set - $|1\rangle$ in the forward sweep set and $|0\rangle$ in the backwards sweep set. This is an inconsistency and implies that the output state $|111\rangle$ is not an output of this circuit with input state $|000\rangle$.

At this point, the values of the intermediate states are already determined. If no inconsistencies were detected in the steps above, the intermediate states are obtained when comparing the intermediate states of each set. If a certain intermediate state is $|0\rangle$ or $|1\rangle$ in, at least, one of the sets, then that is its real state, therefore, the fixed intermediate states of the circuit can be thought of as the union of the sets of intermediate states calculated in the steps above. With the classical intermediate states determined , the amplitudes of all the gates that have classical input states and output states can be computed once and for all. Multiplied together they will be a multiplying factor for the overall amplitude calculation. Assuming the user has chosen the state $|100\rangle$ as output, the comparison and union of the sets of intermediate states calculated in the previous routine is presented in figure 11, with the gates whose amplitude can be computed at this point highlighted.

Finally, an iteration over all possible values of the intermediate states that are $|nd\rangle$ is carried out. For l intermediate states $|nd\rangle$ there would be 2^l possible values, and for each of

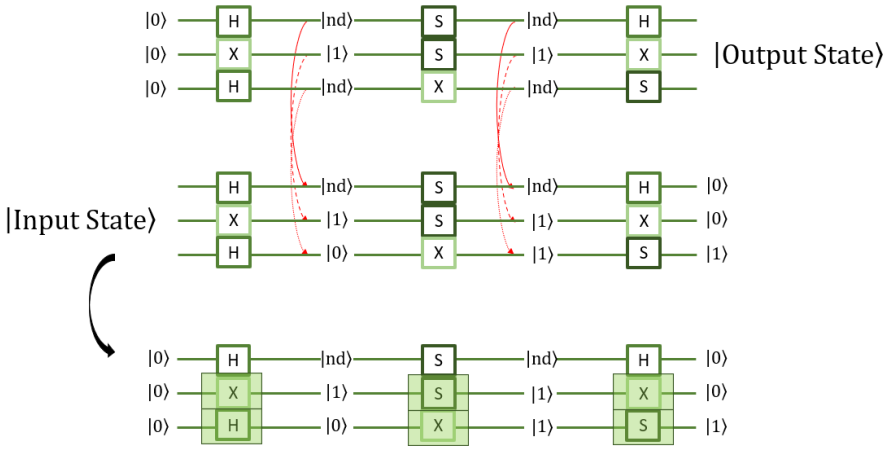


Figure 11: Comparison and union of the sets of intermediate states calculated in the routine that searches for inconsistencies. The gates whose amplitude can be computed at this point are highlighted.

these values it would be necessary to calculate each of the amplitudes of the remaining gates, multiplying them in place with the multiplying factor defined above. After each iteration step, the total amplitude calculated is added to a cumulative variable which, after the final iteration step, will represent the summation shown in eq.(6). The 2^l possible values to be iterated correspond to the paths that are effectively being computed. In this example there are two intermediate states $|nd\rangle$, corresponding to four paths to compute, using this method. These paths are shown in figure 12.

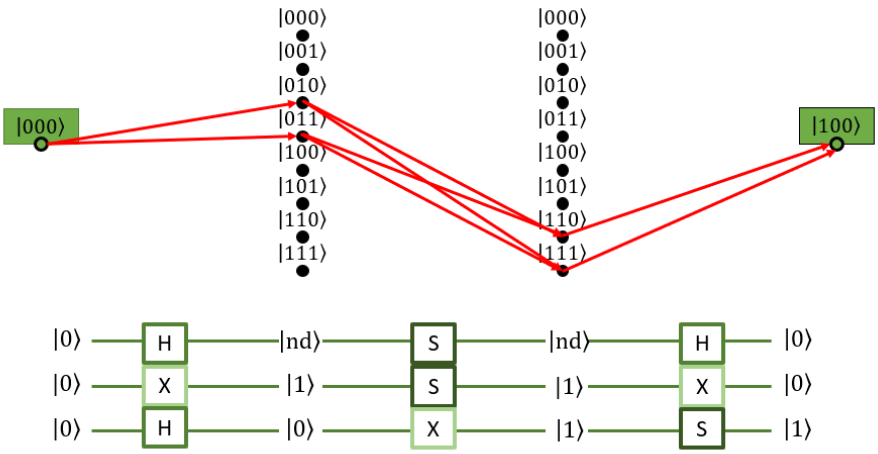


Figure 12: Paths left to compute using the green-red coloring method.

With this algorithmic method and in this example, the number of paths to compute has dropped to four, instead of the canonical version's 64. However, the graph representation of

figure 5 indicates that, in this example, there are only two paths connecting the input state, $|000\rangle$, to the output state, $|100\rangle$, that contribute with non-null probability amplitude.

For a circuit with depth d on n qubits, the worst case scenario for this algorithm is when all intermediate states are not classical. In this case, the probability amplitude of all the gates in the circuit must be calculated for every possible value of the intermediate states. There is a maximum of $n \times d$ gates and $2^{n(d-1)}$ possible values of intermediate states. The time spent pre-processing the circuit, with the coloring sweeps is negligible when compared to the time spent iterating over all possible values for the intermediate states and calculating the amplitudes. Therefore, the time complexity of this algorithm is $O(nd2^{n(d-1)})$. The memory needed in a simulation using this algorithm is linear with the number of qubits and the depth of the circuit, $O(n+d)$.

3.2 PATHNB: PATH SUMS WITH NON-BRANCHING OPTIMIZATION

By inspecting figure 12, it is straightforward to notice that two of the four paths that were a result of applying the method described in section 3.1 have a null probability amplitude. In this example, an S gate has both the input and output state not determined, so an iteration over all possible input/output combinations is carried out. Because of the effect of the S gate:

$$
\begin{aligned}
S &= \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \\
S|0\rangle &= \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle \\
S|1\rangle &= \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = i \begin{bmatrix} 0 \\ 1 \end{bmatrix} = i|1\rangle \\
\langle 0|S|0\rangle &= \langle 0|0\rangle = 1 \\
\langle 0|S|1\rangle &= i\langle 0|1\rangle = 0 \\
\langle 1|S|0\rangle &= \langle 1|0\rangle = 0 \\
\langle 1|S|1\rangle &= i\langle 1|1\rangle = i
\end{aligned}
\tag{13}
$$

the combinations that imply the calculation of the amplitudes $\langle 0|S|1\rangle$ and $\langle 1|S|0\rangle$ represent non-contributing paths and are therefore, a waste of computational time. Taking this into account, an improvement of this method was derived.

The first steps of the *PathNB* algorithm are exactly the same as those detailed in section 3.1: forward and backwards green-red coloring sweeps are performed and, after that, so is the routine that checks for inconsistencies.

In the example presented in figure 12, both the input and output states of an S gate are not classical ($|nd\rangle$). However, if a classical state is assigned to its input state, the output

becomes classical too, because of the non-branching property of the S gate. This suggests that the nature of these intermediate states is not the same: the S gate is a non-branching gate, therefore if the input state is classical, the output state is immediately known and is also classical, while the input state is not classical because of the branching property of the H gate that precedes the S gate. Therefore, a different color scheme could be used to highlight this difference.

After the forward and backwards coloring sweep of the circuit, detailed in 3.1, two different color schemes emerge, as exemplified in figure 7 and figure 8. These two different color schemes can be compared, by performing a logical OR operation between each colored position, where the green positions can be thought of as ones and the red positions as zeros. The result is a coloring scheme that has some positions representing fixed, classical states, colored green, and the others representing non-fixed, not classical states, colored red. Using the same example once again, this process is shown in figure 13.

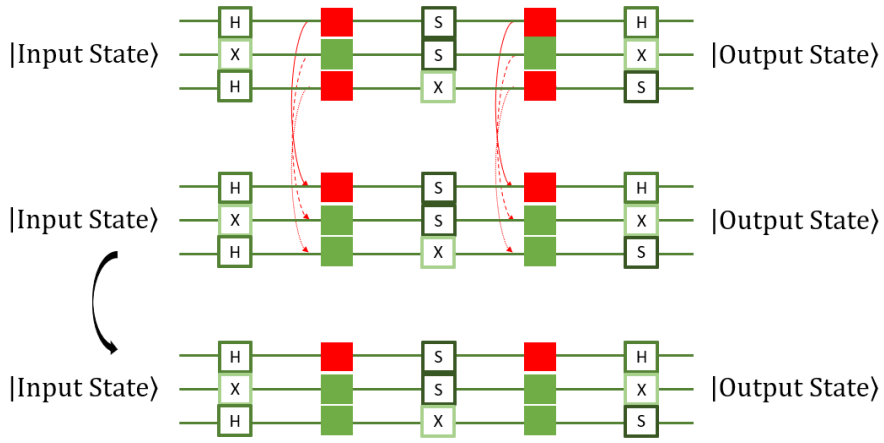


Figure 13: Comparison and result of the logical OR operation between the two color schemes obtained after sweeping the circuit forward and backwards. Positions that are colored green represent fixed, classical states and positions that are colored red represent non-fixed, not classical states.

The next step is to take the resulting color scheme and reprocess it, traversing the circuit from input to output and coloring pink all the positions that represent not classical states (i.e. colored red), and that are immediately after branching gates. All the remaining positions that represent not classical states are colored blue. With these new coloring rules, this coloring scheme is altered. The new, resulting coloring scheme is shown in 14.

After the coloring step, the fixed intermediate states can be determined with the same procedure as they were in section 3.1, but now there are two types of non-classical intermediate states: the ones colored pink and the ones colored blue. The next step is to calculate the amplitudes of all the gates that have both input states and output states determined and to multiply them together to form a multiplying factor for the overall amplitude calculation.

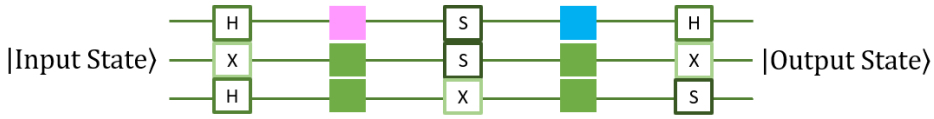


Figure 14: New coloring scheme of the example presented throughout this chapter, after coloring pink all the positions that represent not classical states and that are immediately after branching gates, and coloring blue all the remaining positions that represent not classical states.

Finally, an iteration over all possible values for the intermediate states that are colored pink is carried out. For n intermediate states colored pink there would be 2^n possible values. For each of these values it would be necessary to propagate the certainty, by way of action of a non-branching gate, from the states represented by the positions colored pink to the states represented by the positions colored blue, and calculate each of the amplitudes of the remaining gates, multiplying them in place with the multiplying factor defined above. After each iteration step, the total amplitude calculated is added to a cumulative variable which, after the final iteration step, will represent the summation shown in eq.(6). The 2^n possible values to be iterated correspond to the paths that are effectively being computed. In this example there is one intermediate state colored pink, corresponding to two paths to compute, using this method. These paths are shown in figure 15.

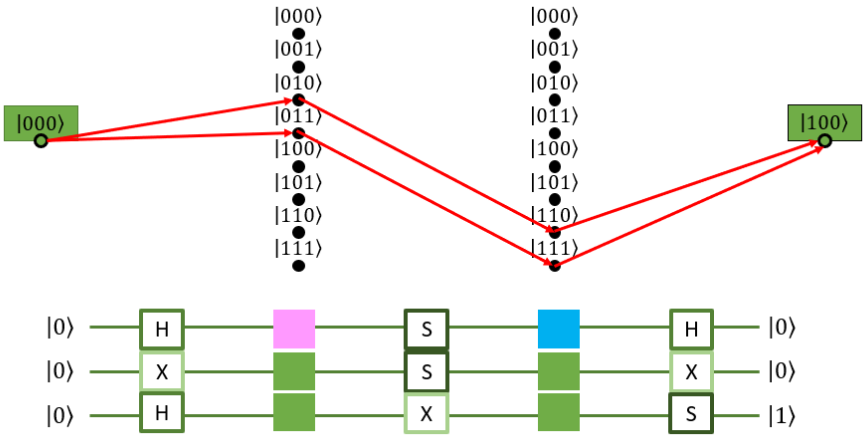


Figure 15: Paths left to compute using the blue-pink coloring method.

With this algorithmic method and in this example, the number of paths to compute has dropped to two, instead of the canonical version's 64, and the four paths of the method described in section 3.1. As expected, the paths left to compute using this method correspond exactly to those highlighted in the graph representation of figure 5.

Any quantum circuit comprised of layers of unitary gates can be written in matrix form. The resulting matrix, U, is unitary

$$U^{\dagger}U = UU^{\dagger} = I. \tag{14}$$

Here, $U^{\dagger}$ is the conjugate transpose of U. Eq.(14) implies that the unitary transformations that occur in such quantum circuits, with an input state $|x\rangle$ and an output state $|y\rangle$ are reversible:

$$\langle y| U |x\rangle = \langle x| U^{\dagger} |y\rangle . \tag{15}$$

The goal in a Feynman path-sum simulation of a quantum circuit is to calculate the amplitude represented by the first term in eq.(15). However, the equality shown in eq.(15) implies that the practical effect of calculating the second term instead is the same, which means that the "temporal" order of the quantum circuit can be reversed without affecting the result of the simulation. This interesting property is taken into account in the *PathNB* algorithm. The pink-blue coloring step described above is done in the input-output direction. But, since the "temporal" order of the circuit is reversible, this step can be done in the output-input direction, as long as, in the amplitude calculation step, the gates are substituted by their conjugate transpose. The utility of considering this direction is that, for some circuits, the resulting number of pink colored positions is different, if the circuit is reversed. Most of the computational time required by this algorithm is spent iterating over the possible state values for the pink positions, and calculating amplitudes. Since, for n intermediate states colored pink there are 2^n possible values to iterate, the computational time is approximately halved for each position that is no longer colored pink, in a given direction. For big enough circuits this difference may have a huge impact in the performance of the algorithm. The circuit presented in figure 16 is an example of a circuit where this phenomenon takes place.

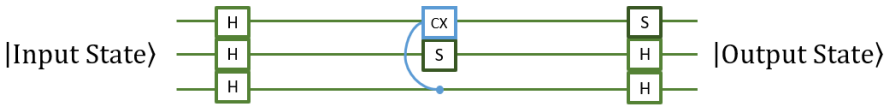


Figure 16: Example of a quantum circuit where the pink-blue coloring in the direction output-input results in a smaller number o pink colored positions.

For a better understanding of this property, the coloring process for this circuit will be presented step by step. The first step is the forward and backwards green-red coloring sweep of the circuit, following the rules shown in figure 6, that can be seen in figure 17. The next step is to compare the two resulting coloring schemes shown in figure 17 by performing the logical OR operation between them. The resulting coloring scheme is presented in

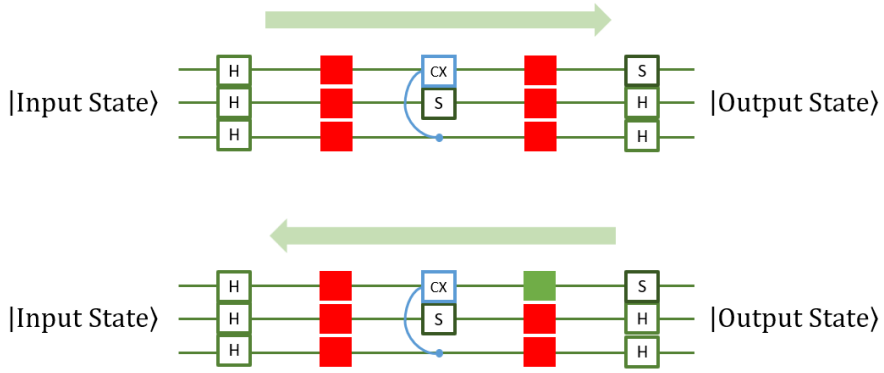


Figure 17: Resulting coloring schemes of the forward and backwards sweep steps applied to the circuit presented in figure 15.

figure 18. Finally the pink-blue coloring can be carried out. Figure 19 shows the result of

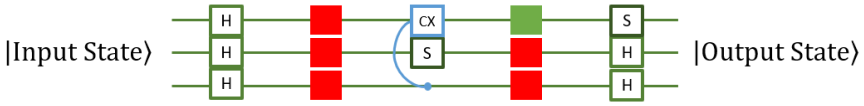


Figure 18: Resulting coloring scheme after performing the logical OR operation between the two coloring schemes depicted in figure 17.

performing this step in both directions. In this circuit, the number of pink colored positions in the input-output direction is larger than the number of pink colored positions in the output-input direction, meaning that, when simulating this circuit, it is advantageous to reverse its "temporal" order, substituting its gates by their conjugate transpose. The CNOT and H gates, used in this circuit, are their own conjugate transpose, while the S gate would be substituted by the $S^\dagger$ gate

$$S^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & -i \end{bmatrix} \tag{16}$$

For a circuit with depth d on n qubits, the worst case scenario for this algorithm is when all intermediate states are not classical and their corresponding position is colored pink. In this case, the probability amplitude of all the gates in the circuit must be calculated for every possible value of the intermediate states. There are is a maximum of nd gates and $2^{n(d-1)}$ possible values of intermediate states. The time spent pre-processing the circuit, with the coloring sweeps is negligible when compared to the time spent iterating over all possible values for the intermediate states and calculating the amplitudes. Therefore, and in this case,

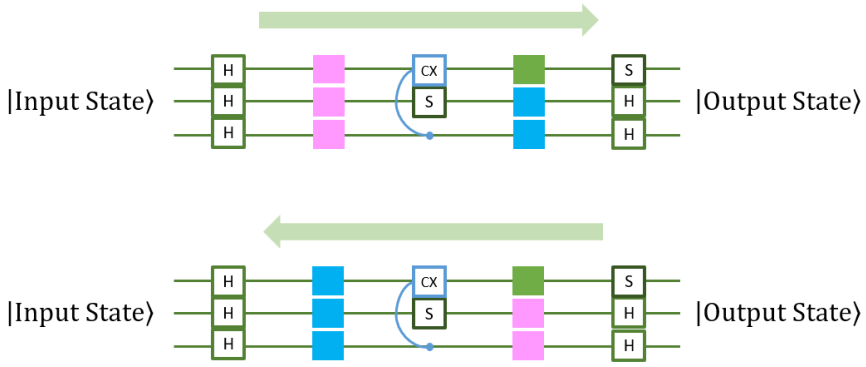


Figure 19: Resulting coloring schemes after performing the pink-blue coloring step in both directions.

the time complexity of this algorithm is $O(nd2^{n(d-1)})$. Although the time complexity for the worst case is the same on this algorithm and in the algorithm described in section 3.1, the time complexity of the typical case scenario is lower in this algorithm, because the number of pink colored positions tends to be lower, and can never be larger, than the number of red colored positions. The memory needed in a simulation using this algorithm is linear with the number of qubits and the depth of the circuit, $O(n + d)$.

3.3 PATHREC: RECURSIVE VERSION OF PATH SUM ALGORITHM

In addition to the *PathNB* algorithm detailed in section 3.2, a recursive Feynman path-sum simulation algorithm that makes use of the coloring process detailed in section3.1 and inspired by the recursive algorithm proposed by Aaronson and Chen (2016) mentioned in section2.3, was also implemented.

For a circuit C, with depth $d > 1$ of layers of unitary gates, the first steps are exactly the same as those in the method described in section 3.1. First, the coloring sweep of circuit, both forward and backwards, then the check for inconsistencies and, finally, the determination of the fixed intermediate states due to certainty propagation from input and output and the action of non-branching gates. The quantum circuit shown in figure 20 will be used as an example throughout this section.

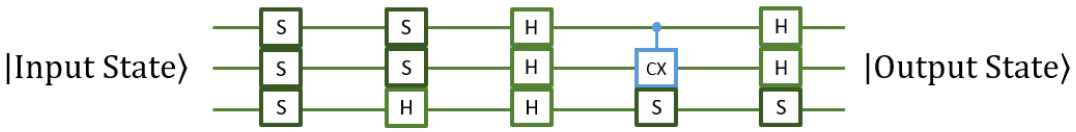


Figure 20: Quantum circuit to be used as an example throughout this section.

Assuming the user has chosen the state $|110\rangle$ as input, and the state $|111\rangle$ as output, after these first steps, some intermediate states are fixed and defined, while others are not. The result of carrying out the steps described above, for this input/output pair, can be seen in figure 21, with the non-defined intermediate states highlighted.

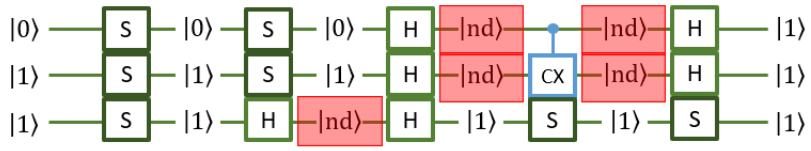


Figure 21: Intermediate states after applying the green-red coloring method.

For a circuit with depth d there are $d-1$ "columns" of intermediate states and it is useful to index them in the input-output direction. The next step of the *PathRec* algorithm is to iterate over all possible values of the non-defined intermediate states of the innermost column of intermediate states and recursively call the algorithm on the two sub-circuits divided by this column of intermediate states, until the base case is reached where $d \leq 2$. In the base case, the total amplitude of the sub-circuit is calculated. If $d = 1$, this probability amplitude calculation is trivial. If $d = 2$, the green-red coloring method is used to find the total probability amplitude. Given a quantum circuit C with depth $d > 2$, an input state $|x\rangle$ and an output state $|y\rangle$ the following equation represents the recursive calls of the algorithm:

$$\langle y| C |x\rangle = \sum_{z \in (0,1)^n} \langle y| C_2 |z\rangle \cdot \langle z| C_1 |x\rangle \tag{17}$$

Here, $|z\rangle$ is the intermediate state of the innermost column of intermediate states, C_1 is the sub-circuit delimited by the input state $|x\rangle$ and the intermediate state $|z\rangle$, and C_2 is the sub-circuit delimited by the intermediate state $|z\rangle$ and the output state $|y\rangle$. In the specific case of this example, the innermost column of intermediate states is the second one. Figure 21 shows that this column has one non-defined state, meaning that there are two possible values for the intermediate state $|z\rangle$ to iterate through. Figure 22 shows how the circuit is divided, and the resulting sub-circuits, C_1 and C_2, in the first iterative step and level of recursion.

The sub-circuit C_1 has depth $d = 2$, so it corresponds to the base case. The probability amplitude $\langle z| C_1 |x\rangle$ is calculated at once using the green-red method and stored. On the contrary, the sub-circuit C_2 has depth $d = 3$ and, therefore, the steps to determine its intermediate states are carried out and, afterwards, there is another recursive call of the algorithm, dividing this sub-circuit into two other sub-circuits. An illustration of this process

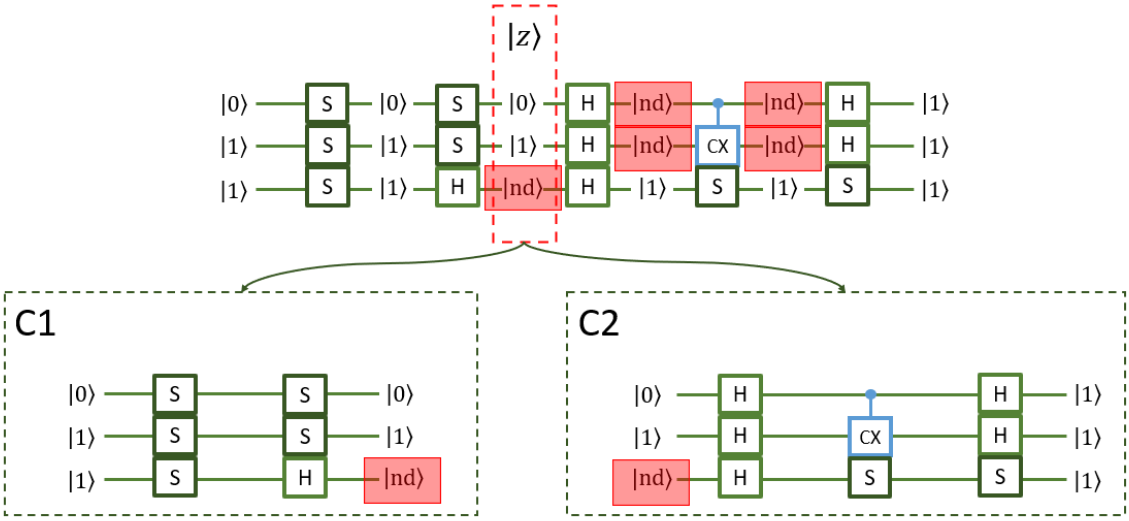


Figure 22: Division of the original circuit into two sub-circuits, C_1 and C_2, in the first iterative step and level of recursion.

can be seen in figure 23. Both resulting sub-circuits, $C_{2,1}$ and $C_{2,2}$, correspond to base cases. The sub-circuit $C_{2,1}$ has depth $d = 1$, so its corresponding probability amplitude is calculated trivially and stored. The sub-circuit $C_{2,2}$ has depth $d = 2$ and, therefore, its corresponding probability amplitude is calculated using the green-red method and multiplied by the probability amplitude calculated with the sub-circuit $C_{2,1}$. The sub-circuit C_2 has two non-defined states in the intermediate state $|z\rangle$, meaning that there are four possible values of $|z\rangle$ to iterate through, generating four probability amplitude values to be summed over and, then, multiplied by the probability amplitude calculated with the sub-circuit C_1. Then, going back to the division the original circuit, where it was noticed that the innermost intermediate state, $|z\rangle$, had two possible values to iterate through, this whole process is repeated for the second of these values. In the end, after all the sums and multiplications of probability amplitudes, the resulting amplitude corresponds to $\langle y| C |x\rangle$.

For a circuit with depth d on n qubits, the worst case scenario for this algorithm is when all intermediate states are not classical and their corresponding position is colored red. In this case, the intermediate state, $|z\rangle$, of every circuit in every level of recursion is completely non-classical. Figure 24 shows that there is a maximum of $log_2(d)$ levels of recursion and $d/2$ sub-circuits in the last level of recursion. In the original circuit, in level 1, there are n non-classical intermediate states and 2^n possible values to iterate through. For each of these 2^n possible values, there are 2^n possible values to iterate through in each of the two circuits in level 2. At this point there are $2 \times 2^n \times 2^n$ possible values to iterate through. Carrying out this procedure through to the last level of recursion results in $d/2 \times 2^{nlog_2(d)}$ possible values to iterate through. For each of these values, the probability amplitude of all the $2n$ gates of every sub-circuit in the last level of recursion must be calculated.

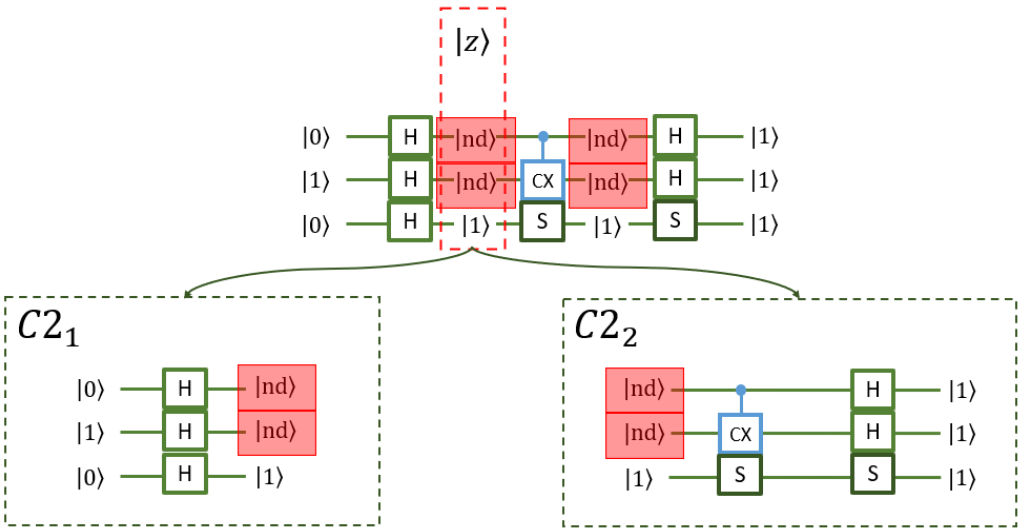


Figure 23: Division of the C_2 circuit into two sub-circuits, C_{2_1} and C_{2_2}, using the first value of the non-defined intermediate state.

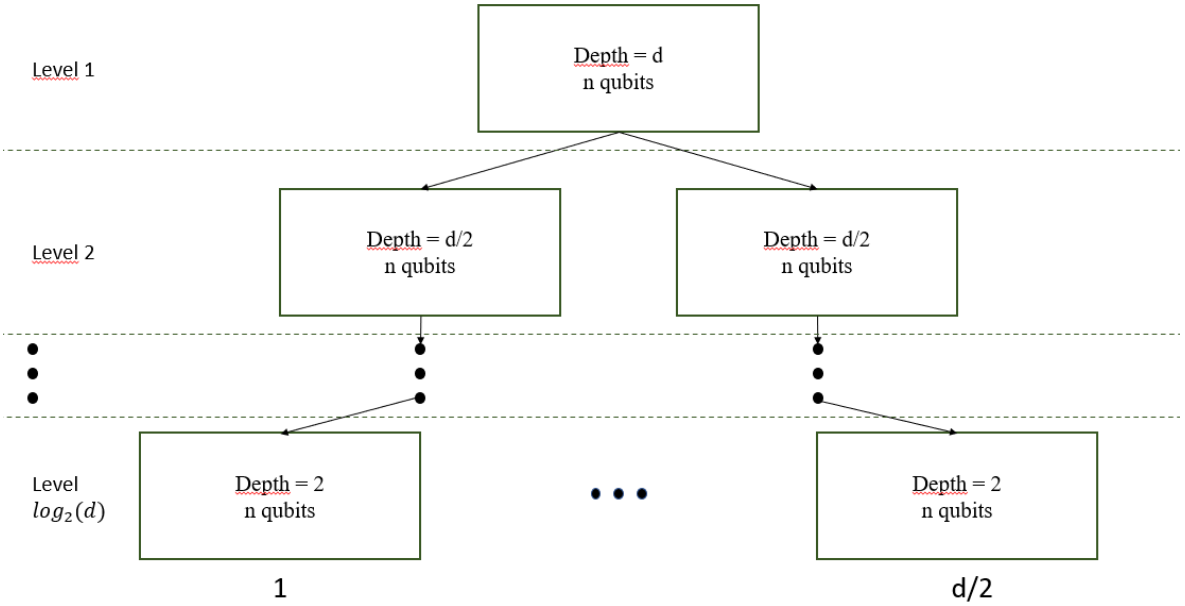


Figure 24: Levels of recursion of the *PathRec* algorithm.

The time spent pre-processing the circuit, with the coloring sweeps is negligible when compared to the time spent iterating over all possible values for the intermediate states and calculating the amplitudes. Therefore, and in this case, the time complexity of this algorithm is $O(2n \times d/2 \times 2^{nlog_2(d)})$, or $O(nd2^{nlog_2(d)})$. This time complexity is lower than *PathNB's*, $O(nd2^{n(d-1)})$, because $nlog_2(d) < n(d-1)$, for every value of n and $d > 1$. The memory needed in a simulation using this algorithm is linear with the number of qubits, $O(n)$.

3.4 PRE-COMPILATION

Most of the computational time running a simulation in both algorithms is spent iterating over all possible values of the non-defined intermediate states and calculating amplitudes. The numbers of iterations to carry out scales exponentially with the number of non-defined intermediate states which, in turn, are related to the circuit structure. Some quantum gate identities can be incorporated in both algorithms in a pre-compilation step to change the structure of the circuit, reducing the number of non-defined intermediate states, and making the circuit simulation more efficient.

The H gate matrix is unitary

$$H^\dagger H = HH^\dagger = I \tag{18}$$

and hermitian

$$H^\dagger = H \tag{19}$$

therefore, combining equations eq.(18) and eq.(19) results in the following identity.

$$HH = I \tag{20}$$

This identity can be particularly useful for circuits with a large number of H gates. Considering the extreme example of a quantum circuit, C, on n qubits and with depth d of layers of H gates and the computational basis states $|x\rangle$ and $|y\rangle$, the green-red and pink blue coloring methods described in sections 3.1 and 3.2, respectively, would result in a number $n \times (d-1)$ of non-defined intermediate states. This corresponds to the worst case scenario for both *PathNB* and *PathRec*, because all the intermediate states would be non-defined, and the number of possible values to iterate through, $2^{n(d-1)}$, is the maximum for a circuit with this configuration. If, before carrying out the simulation, a pre-compilation of the circuit was performed, applying the identity shown in eq.(20), C would be transformed into a new circuit, with either $d/2$ layers of identity (I) gates, in the case where d is even, or $(d-1)/2$ layers of I gates and a layer of H gates, in the case where d is odd. The layers of I gates can be removed without affecting the probability amplitude $\langle y| C |x\rangle$, meaning that, if d is even, there would be no circuit at all, and, if d is odd, the circuit would be just a layer of H gates. Therefore, the probability amplitude $\langle y| C |x\rangle$ could be trivially computed in $O(1)$, if d is even, or $O(n)$ time, if d is odd, with both algorithms.

Likewise, other identities that reduce the H-count can be incorporated. Figure 25 shows some identities to be incorporated in a pre-compilation step.

Considering again the circuit used as an example throughout section 3.3, shown in figure 20. Applying a pre-compilation to this circuit results in a transformation that can be seen in figure 26. The resulting circuit is classical and can be simulated trivially.

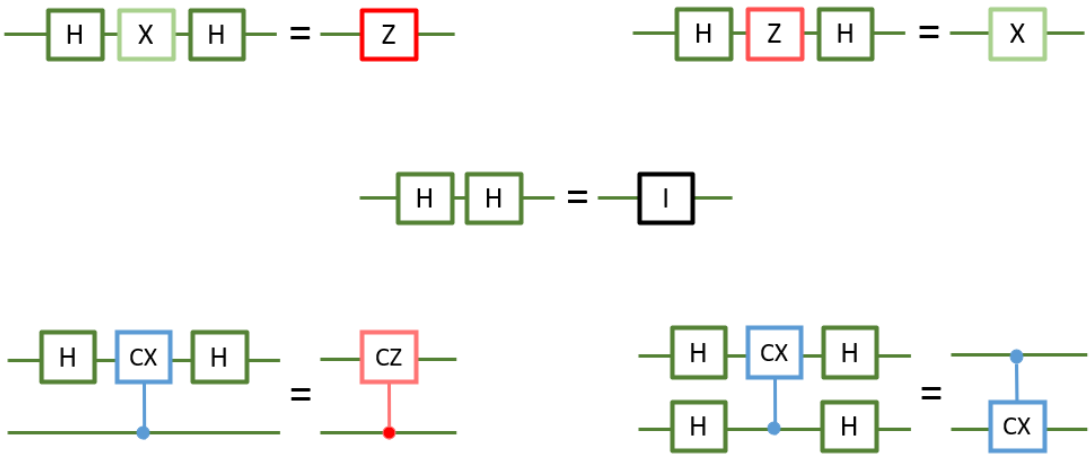


Figure 25: Quantum gate identities to incorporate in a pre-compilation step.

The identification and application of the quantum gate identities shown in figure 25 are a rather simplistic approach to circuit optimization and compilation, although the result of their application might yield a considerable speedup in the simulation of quantum circuits using the *PathNB* and *PathRec* algorithms. For a more profound knowledge on how to reduce the H-count, Abdessaied et al. (2014) introduces a scheme to optimize the T-depth of quantum circuits based on H gate reductions. Reducing the T-depth and T-count holds special interest, because implementing T gates is expensive (Mooney et al. (2021)), and recent research is aiming at minimizing the use of such gates. Heyfron and Campbell (2018) developed a circuit compiler and benchmarked it on random circuits, from which they determined that it yields the lowest T counts on average. They also benchmarked it on a library of reversible logic circuits that appear in quantum algorithms and found that it reduced the T count for 97% of the circuits with an average T-count saving of 20% when compared against the best of all previous circuit decompositions.

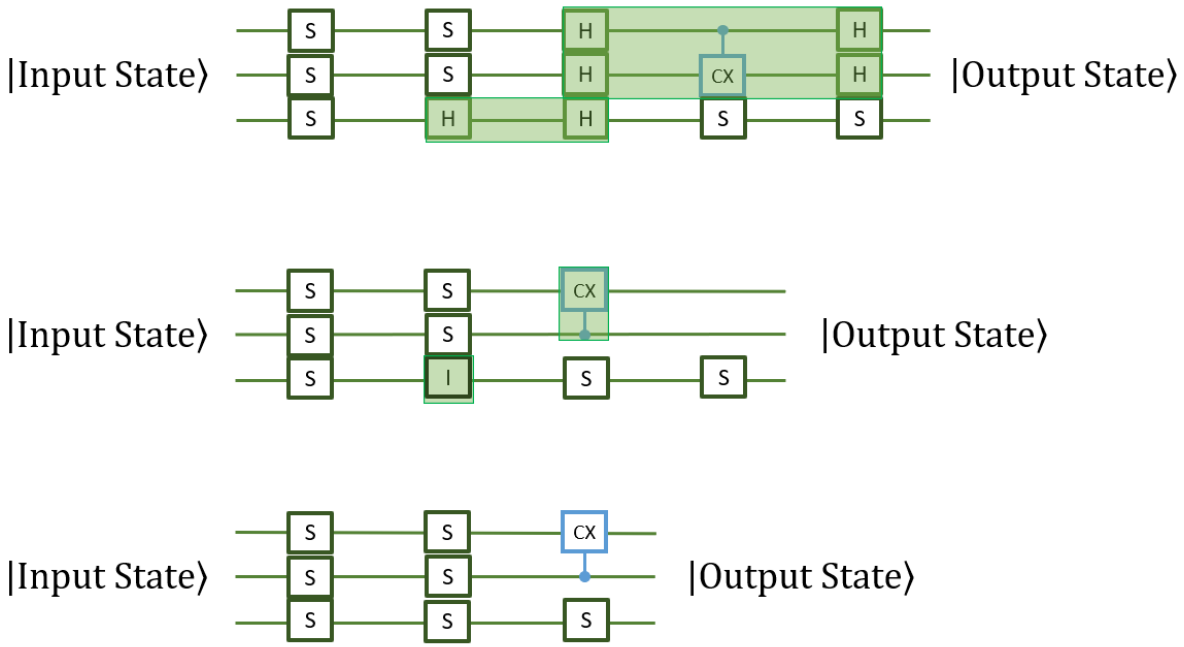


Figure 26: Pre-compilation step for the circuit presented in figure 19.

4

EXPERIMENTAL RESULTS

Based on the simulation algorithms presented in chapter 3, two classical simulators of quantum computers were implemented, *SimulatorNB* and *SimulatorRec*. All the code, CSV files with test results and circuits tested are made publicly available in Github[1]. This chapter is dedicated to the experimental process regarding the performance comparison tests made using both simulators. Section 4.1 features the descriptions of the gate set accepted by both simulators, the ensemble of circuits on which experimental tests were conducted, the pre-compilation step implemented, and the metrics measured and used in each experiment. This section is concluded with the presentation of the pseudocode of both simulators. Throughout section 4.2 several experimental results are presented regarding the correctness verification of both simulators, the effect of the pre-compilation step on the structure of different circuit types from the ensemble of circuits used and, finally, the performance comparison between both simulators. Tests were made for different types of quantum circuits, culminating in the presentation of results of the performance comparison when simulating circuits where a Schrödinger-type simulator would consume excessive amounts of memory.

4.1 EXPERIMENTAL SETUP AND IMPLEMENTATION

Based on the algorithms *PathNB* and *PathRec* presented in sections 3.2 and 3.3, respectively, two classical quantum simulators designed for simulating quantum circuits were implemented, *SimulatorNB* and *SimulatorRec*. Both simulators were implemented in Python 3.9.0. and designed to simulate quantum circuits from Qiskit.

Originally the set of gates was composed only by the generators of the Clifford group - H, CNOT and S gates - augmented with the T gate, which is known to be an universal set of gates, meaning that any other unitary operation can be approximated, with arbitrarily high precision, with a sequence of gates from this set. However, to take into account the quantum gate identity shown in eq.(20) in a pre-compilation step, this set was augmented with the identity gate. These simulators compute the probability amplitude $\langle y| C |x\rangle$ for circuits, C,

1 https://github.com/DavidACFerreira/Feynman-Path-sum-Classical-Quantum-Simulator

constituted by this set of gates, input state, $|x\rangle$, in the computational basis, and output state, $|y\rangle$, in the computational basis. The full set of gates supported by both simulators is shown in Table 1.

Gate	Matrix
H	$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$
S	$\begin{bmatrix}1 & 0\\ 0 & -i\end{bmatrix}$
CNOT	$\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{bmatrix}$
T	$\begin{bmatrix}1 & 0\\ 0 & e^{i\pi 4}\end{bmatrix}$
I	$\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$

Table 1: Set of gates supported by *SimulatorNB* and *SimulatorRec*

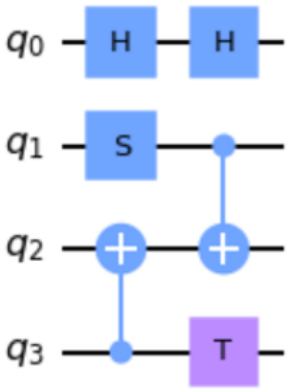
The ensemble of circuits used to test both simulators consisted of randomly generated Qiskit quantum circuits with the set of gates limited to that in table 1. Random circuits are often used in benchmarking classical simulation, since some other benchmark circuits use more elaborate gates not included in our set of gates. This ensemble of circuits is defined by three variables - i) the number of qubits, n, ii) the depth of the circuit, d, and iii) the probability of a given gate in the circuit being the H gate, *h_probability*. The third variable was defined to study the influence of the number of branching gates on the time the simulators take to compute $\langle y| C |x\rangle$. The probability of a given gate in the circuit being one of the remaining gates from the set is the same for every gate type and its value is $(1 - h\_probability)/4$. The depth corresponds to the number of unitary layers of gates acting on n qubits. A circuit C of this ensemble is built by probabilistically choosing the gates in accordance with the *h_probability*, and the probability defined above for the remaining gates, for each layer of the circuit. The qubits each of the chosen gates act on are chosen in a random, uniform way, over all the qubits that still haven't been assigned to a gate. Control conditions were implemented to make it impossible to choose a 2-qubit gate in the case where there is only one qubit that still has not been assigned to a gate.

After randomly generating a Qiskit quantum circuit, *qc*, a user can carry out a simulation for any n-sized computational basis *input_state* and *output_state*. For a quantum circuit acting on n qubits, the input and output computational basis states are merely n-sized binary vectors. Other important data structures that are used during a simulation, in both simulators, include:

- The list of quantum operations that represents the circuit, *qc.data*. It is generated using an attribute of Qiskit's quantum circuits, QuantumCircuit.data. Each element of this list is a list corresponding to each layer of gates, ordered in the input-output direction. This list has information about the gates that appear in the circuit, the qubits they act on and the order in which they appear. Figure 27 shows the *qc.data* for a Qiskit quantum circuit.

- The list that contains the intermediate states, *inter_states*. For a circuit on n qubits with depth d, this list will have $d-1$ n-sized vectors each containing the state of the qubits after a given layer of gates, ordered in the input-output direction.

- The lists that contain color schemes, such as *green_red* and *blue_pink*. For a circuit on n qubits with depth d, these lists will have the same structure as the *inter_states* list, but their n-sized vectors will contain the color code of each qubit instead.




```
[[['H', 0], ['S', 1], ['CX', 3, 2]], [['H', 0], ['CX', 1, 2], ['T', 3]]]
```

Figure 27: A Qiskit quantum circuit and its corresponding list of quantum operations.

The pre-compilation step is done by searching for two consecutive H gates acting on the same qubit and substituting them by an I gate. For the quantum circuit presented in figure 27, the effect of the pre-compilation step is shown in figure 28.

Several performance tests were made on various input - n, d and *h_probability* - combinations, for both simulators. Each circuit was simulated before and after the pre-compilation step, to study the effect of this step on the performance of both simulators. In each simulation various parameters were measured. These parameters include:

- The percentage of H gates in the circuit, since the H gate is the only branching gate in the set of gates supported by both simulators, and therefore, the number of H gates in

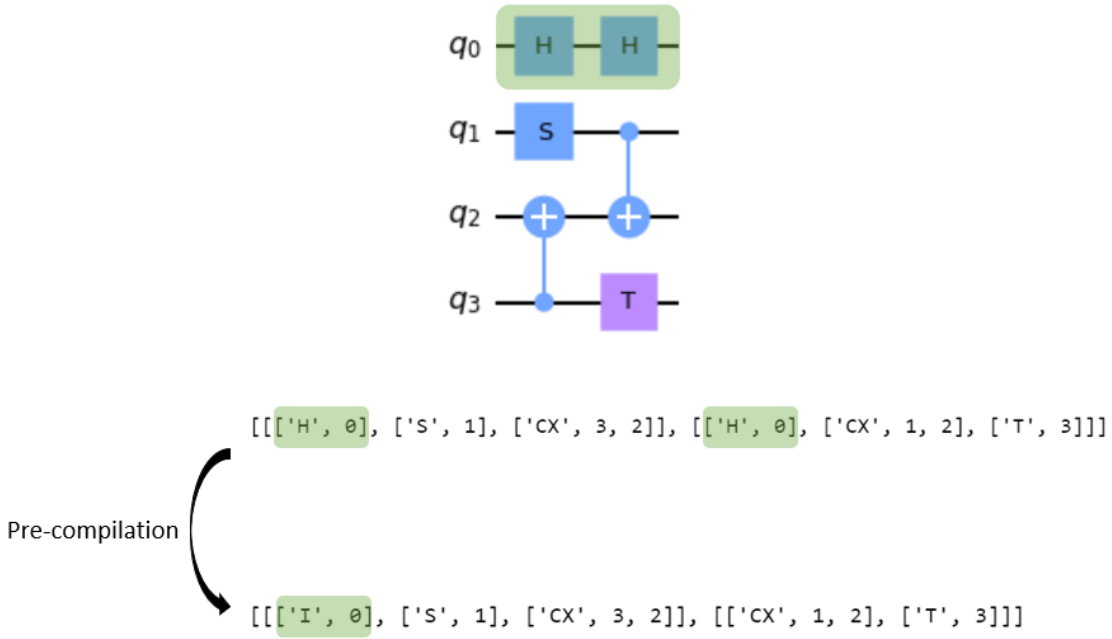


Figure 28: Effect of the pre-compilation step for the quantum circuit presented in figure 27.

a circuit is related with the number of paths to compute and the time complexity of the simulation;

- The number of red-colored intermediate states;
- The number of pink-colored intermediate states, to study the effect of the blue-pink coloring method on the number of paths to compute and on the time complexity of a simulation using the *SimulatorNB*;
- The runtime of a simulation;
- The memory usage of a simulation;
- The output of the simulation, i.e., the amplitude $\langle y| C |x\rangle$, to compare it with the expected output, to validate the correctness of both simulators.

Finally the pseudocode for *SimulatorNB* and *SimulatorRec* is presented.

Algorithm 1: SimulatorNB

procedure SimulatorNB($qc.data$, $input\_state$, $output\_state$, $pre\_compilation\_flag$)
  if $pre\_compilation\_flag$ **then**
    $qc.data \leftarrow$pre_compilation($qc.data$)
  end if
  $depth \leftarrow$len($qc.data$)
  $n \leftarrow$len($input\_state$)
  $fs \leftarrow$forwardSweep($qc.data$)
  $bs \leftarrow$backwardsSweep($qc.data$)
  if inconsistencyCheck($qc.data, input\_state, output\_state, fs, bs$) **then**
    return o
  end if
  $green_red \leftarrow$greenRedColoring($qc.data, fs, bs$)
  if n_reds **in** $green\_red = 0$ **then**
    $inter_states \leftarrow$interStatesCalc($qc.data, input\_state, output\_state, green\_red$)
    return amplitude($qc.data, input\_state, output\_state, inter\_states$)
  end if
  $blue_pink \leftarrow$bluePinkColoring($qc.data, green\_red$)
  $inter_states \leftarrow$interStatesCalc($qc.data, input\_state, output\_state, blue\_pink$)
  $amp\_final \leftarrow 0$
  $mult_factor \leftarrow$amplitude($qc.data, input\_state, output\_state, inter\_states$)
  for $i \leftarrow 0$ to $2^{n\_pinks} - 1$ **do**
    $bitstring \leftarrow$ bin(i, n_pinks) ▷ n_pinks-sized binary representation of i
    $bitstring\_index \leftarrow 0$
    $mult\_aux \leftarrow 1$
    $all\_states \leftarrow input\_state + inter\_states + output\_state$
    for $j \leftarrow 0$ to $depth - 1$ **do**
      for $gate$ in $qc.data[j]$ **do**
        if $gate[0] = H$ and $all\_states[j+1][gate[1]] = pink$ **then**
          $all\_states[j+1][gate[1]] \leftarrow bitstring[bitstring\_index]$
          $bitstring\_index \mathrel{+}= 1$
          $mult_aux \mathrel{+}=$Hgate$[all_states[j][gate[1]]][all_states[j+1][gate[1]]]$
        end if
        if $gate[0] = H$ and $j > 0$ and $blue\_pink[j-1][gate[1]] \neq green$ **then**
          $mult_aux \mathrel{+}=$Hgate$[all_states[j][gate[1]]][all_states[j+1][gate[1]]]$
        end if
        if $gate[0] = CX$ and $all\_states[j+1][gate[1]] = blue$ **then**

$all\_states[j+1][gate[1]] \leftarrow all\_states[j][gate[1]]$
$mult\_aux$ +=CXgate$[(all_states[j][gate[1]], all_states[j][gate[2]])][(all_states[j+1][gate[1]], all_states[j+1][gate[2]])]$
end if
if $gate[0] = CX$ and $all\_states[j+1][gate[2]] = blue$ **then**
$all\_states[j+1][gate[2]] \leftarrow |all\_states[j][gate[2]] - all\_states[j][gate[1]]|$
$mult\_aux$ +=CXgate$[(all_states[j][gate[1]], all_states[j][gate[2]])][(all_states[j+1][gate[1]], all_states[j+1][gate[2]])]$
end if
if $gate[0] = S$ and $all\_states[j+1][gate[1]] = blue$ **then**
$all\_states[j+1][gate[1]] \leftarrow all\_states[j][gate[1]]$
$mult\_aux$ +=Sgate$[all_states[j][gate[1]]][all_states[j+1][gate[1]]]$
end if
if $gate[0] = T$ and $all\_states[j+1][gate[1]] = blue$ **then**
$all\_states[j+1][gate[1]] \leftarrow all\_states[j][gate[1]]$
$mult\_aux$ +=Tgate$[all_states[j][gate[1]]][all_states[j+1][gate[1]]]$
end if
if $gate[0] = I$ and $all\_states[j+1][gate[1]] = blue$ **then**
$all\_states[j+1][gate[1]] \leftarrow all\_states[j][gate[1]]$
$mult\_aux$ +=Igate$[all_states[j][gate[1]]][all_states[j+1][gate[1]]]$
end if
end for
end for
$mult\_aux *= mult\_factor$
amp_final += $mult\_aux$
end for
return amp_final
end procedure

Algorithm 2: SimulatorRec

procedure SimulatorRec($qc.data$, $input\_state$, $output\_state$, $pre\_compilation\_flag$)
  if $pre\_compilation\_flag$ **then**
    $qc.data \leftarrow$pre_compilation($qc.data$)
  end if
  $depth \leftarrow$len($qc.data$)
  $n \leftarrow$len($input\_state$)
  if $depth = 1$ **then**
    return amplitude($qc.data, input\_state, output\_state$)
  end if
  $fs \leftarrow$forwardSweep($qc.data$)
  $bs \leftarrow$backwardsSweep($qc.data$)
  if inconsistencyCheck($qc.data, input\_state, output\_state, fs, bs$) **then**
    return o
  end if
  $green_red \leftarrow$greenRedColoring($qc.data, fs, bs$)
  $inter_states \leftarrow$interStatesCalc($qc.data, input\_state, output\_state, green\_red$)
  if n_reds **in** $green\_red = 0$ **then**
    return amplitude($qc.data, input\_state, output\_state, inter\_states$)
  end if
  $all\_states \leftarrow input\_state + inter\_states + output\_state$
  if $depth > 2$ **then**
    for $i \leftarrow 0$ to n **do**
      if $green\_red$[len($green\_red/2$)][i]$= red$ **then**
        n_reds_mid += 1
      end if
    end for
    $amp\_final \leftarrow 0$
    for $i \leftarrow 0$ to $2^{n\_reds\_mid} - 1$ **do**
      $bitstring \leftarrow$ bin(i, n_reds_mid) ▷ n_reds_mid-sized binary representation of i
      $bitstring\_index \leftarrow 0$
      for $i \leftarrow 0$ to n **do**
        if all_states[len($all\_states/2$)][i]$= red$ **then**
          all_states[len($all\_states/2$)][i] $\leftarrow bitstring$[$bitstring\_index$]
        end if
      end for
      $amp\_final \leftarrow amp\_final$ ▷ Recursive call
      + SimulatorRec($qc.data$[$0 : depth/2$], all_states[0], all_states[(len(all_states)-1)])

```
            + SimulatorRec(qc.data[depth/2 : depth], all_states[(len(all_states)−1)/2],
            all_states[len(all_states)−1])
        end for
    end if
    if depth ≤ 2 then
        amp_final ← 0
        for i ← 0 to 2^n_reds − 1 do
            inter_states_aux ← inter_states
            bitstring ← bin(i, n_reds)              ▷ n_reds-sized binary representation of i
            bitstring_index ← 0
            for j ← 0 to len(inter_states_aux) − 1 do
                for state in inter_states_aux[j] do
                    if state = red then
                        state ← bitstring[bitstring_index]
                        bitstring_index += 1
                    end if
                end for
            end for
            amp_final += amplitude(qc.data, input_state, output_state, inter_states_aux)
        end for
    end if
    return amp_final
end procedure
```

4.2 RESULTS

In this section, the results regarding the tests of validation and performance comparison of *SimulatorNB* and *SimulatorRec*, with and without the pre-compilation step, are presented, with special focus on the time and memory usage. The results that follow were all obtained using Python 3.9.0. in a single workstation. For reference the machine used has an 8-core AMD Ryzen 7 4700U processor @ 2 GHz - 4.1 GHz and 16GB RAM.

4.2.1 *Correctness Validation*

The metric used to validate the algorithms of both simulators was the total variation distance (TVD), which is commonly used to to measure the distance between two probability

distributions. Formally, given two discrete probability distributions P and Q over the same sample space X, the TVD may be defined as:

$$TVD(P,Q) = \frac{1}{2}\sum_{x}|P(x) - Q(x)| \tag{21}$$

where the sum is taken over all possible outcomes $x \in X$. In the context of the task of validating the correctness of both simulators, the TVD may be used to compare the output probabilities of the implemented simulators with the expected probabilities obtained from a trusted quantum computing library. If the TVD between the two probability distributions is below a certain threshold, ϵ, it is considered that the simulator algorithm produces correct results.

To this end, tests over all possible combinations of computational basis input and output states for several quantum circuits were carried out. For each quantum circuit and each possible input state, the TVD between the output distribution obtained from our simulators and the expected output distribution obtained from the Statevector Simulator, from Qiskit's *BasicAer* module, was calculated. The resulting TVD values were then analyzed to determine whether they met a predefined threshold for correctness.

As the number of qubits in a quantum circuit grows, the number of possible input and output states also increases exponentially. For a circuit on n qubits, there are 2^n possible input states and 2^n possible output states, resulting in a total of $(2^n)^2 = 2^{2n}$ possible combinations of input and output states to consider. This can quickly become computationally infeasible for circuits with a large number of qubits. Furthermore, a circuit with a large depth is bound to have a larger number of H gates, resulting in an also large number of paths to compute, for each of the 2^{2n} possible combinations of input and output states. Considering this a choice was made to run these tests on circuits acting on a maximum of 8 qubits and with a maximum depth $d = 8$. Given these restrictions, a reasonable value for the threshold, ϵ, was defined to be $\epsilon = 1 \times 10^{-12}$, compatible with the expected numerical round-off error.

As the behaviour of both simulators is heavily influenced by the number of H gates in the circuit, the first tests were run on circuits from the ensemble of circuits described in section 4.1, on 4 qubits, with depth $d = 4$ and with varying H gate probability. This input parameter was forced to vary between the limit cases, $h\_probability = 0$, where there are no H gates in a circuit, and $h\_probability = 1$, where all the gates in a circuit are H gates. The step was chosen to be 0.25, which holds five values of $h\_probability$: $0, 0.25, 0.5, 0.75$ and 1. For each of these values, 20 different circuits were simulated, resulting in a total of 100 circuits. Each of these was simulated using both simulators, with and without the pre-compilation step. For each of these simulations, the TVDs associated with each possible computational basis input state were calculated and compared with ϵ. Table 2 summarizes these results, presenting the TVD with highest value calculated for both simulators, before and after the pre-compilation

step. By inspecting table 2 it is clear that the TVD with highest value calculated in this set of tests is, by a comfortable margin, smaller than the threshold value ϵ.

	SimulatorNB	SimulatorNB PC	SimulatorRec	SimulatorRec PC
Highest TVD	3.23×10^{-15}	2.25×10^{-15}	3.08×10^{-15}	2.25×10^{-15}
% of H gates	80.00	26.67	80.00	26.67

Table 2: Highest total variation distance (TVD) calculated for *SimulatorNB* and *SimulatorRec*, after tests on 100 different random circuits on 4 qubits, with depth $d = 4$ and with *h_probability* varying between 0 and 1. Each circuit was also simulated by both simulators after the pre-compilation step (PC).

To verify the correctness of both simulators for a varying number of qubits and circuit depth, tests were run on random circuits with the number of qubits and the depth of the circuit varying between 2 and 8. The *h_probability* was fixed with a value $h\_probability = 0.05$ to limit the runtime of these tests. With these restrictions, for each possible pair (n, d), ten different circuits were simulated, resulting in a total of 490 circuits. Each of these was simulated using both simulators, with and without the pre-compilation step. For each of these simulations, the TVDs associated with each possible computational basis input state were calculated and compared with ϵ. Table 3 summarizes these results, presenting the TVD with highest value calculated for both simulators, before and after the pre-compilation step. By inspecting table 3 it is clear that the TVD with highest value calculated in this set of tests is, once again, by a comfortable margin, smaller than the threshold value ϵ.

	SimulatorNB	SimulatorNB PC	SimulatorRec	SimulatorRec PC
Highest TVD	2.25×10^{-15}	2.25×10^{-15}	2.25×10^{-15}	2.25×10^{-15}
# qubits	5	5	5	5
depth	6	6	6	6

Table 3: Highest total variation distance (TVD) calculated for *SimulatorNB* and *SimulatorRec*, after tests on 490 different random circuits with $h\_probability = 0.05$, with the number of qubits $2 \leq n \leq 8$ and with depth $2 \leq d \leq 8$. Each circuit was also simulated by both simulators after the pre-compilation step (PC).

The results shown in tables 2 and 3 present a fairly rigorous validation of the correctness of both quantum simulators. The TVD tests demonstrated that the behaviour of *SimulatorNB* and *SimulatorRec* is in line with what was predicted, with all TVD values falling below the chosen threshold, ϵ, for correctness. This provides strong evidence for the validity and reliability of the simulators, and gives us confidence in its ability to accurately simulate the behaviour of any quantum circuit from the ensemble defined in section 4.1.

4.2.2 *Effect of the pre-compilation step*

Next, the impact of the pre-compilation step on the structure of random circuits is examined. As mentioned in section 3.4, the pre-compilation step aims to optimize the circuit structure, which can significantly speed up the simulation process. In the context of our simulators, the goal of the pre-compilation step is to reduce the H count. In chapter 3 the concept of colored intermediate states was introduced. It was concluded that the runtime of a simulation, using any of the implemented simulators, is predominantly determined by the number of intermediate states colored red, or pink. To put it simply, from now on, the number of intermediate states colored red after pre-processing a quantum circuit will be referred to as n_reds and the number of intermediate states colored pink as n_pinks. Tests were made in which, for each circuit, the percentages of H gates, of intermediate states colored red, and of intermediate states colored pink were measured, before and after the pre-processing step. These last two percentages are calculated as follows:

$$\%reds = \frac{n\_reds}{n(d-1)} \times 100 \tag{22a}$$

$$\%pinks = \frac{n\_pinks}{n(d-1)} \times 100 \tag{22b}$$

while the percentage of H gates is simply:

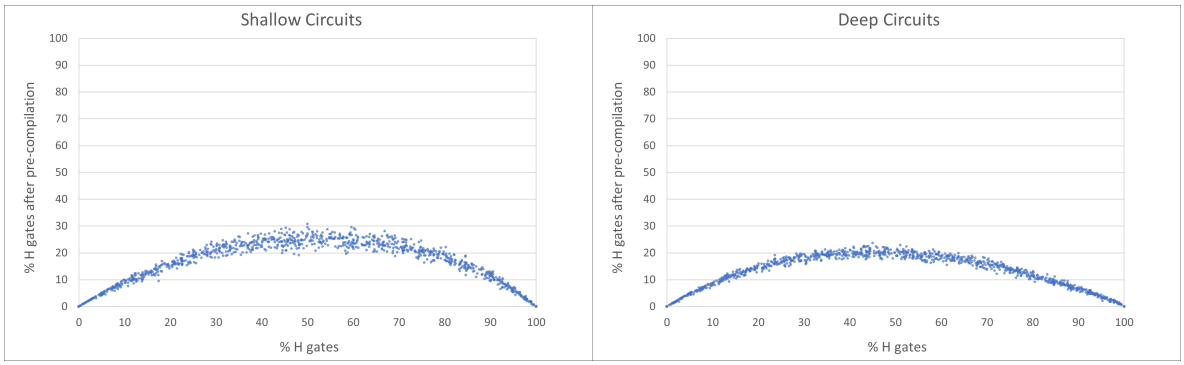
$$\%Hgates = \frac{n\_Hgates}{n\_gates} \times 100 \tag{23}$$

where n_Hgates and n_gates are the number of H gates and the total number of gates in a circuit, respectively. These tests were run on two types of random circuits:

- shallow circuits with a large number of qubits - with $d = 4$ and $n = 100$;
- deep circuits with a small number of qubits - with $d = 100$ and $n = 4$.

In this way one is allowed to assess the characteristics of the circuit structure as a function of d and n. For each of these random circuit types, the input parameter $h\_probability$ was forced to vary between the limit cases, $h\_probability = 0$ and $h\_probability = 1$. The step was chosen to be 0.01, which holds 101 values of $h\_probability$. For each of these values, 10 different circuits were simulated, resulting in a total of 1010 circuits.

Figure 29 presents the results concerning the effect of the pre-compilation step on the percentage of H gates in a circuit. By inspecting the graphs of figure 29, one can see the effectiveness of the pre-compilation approach in reducing the number of H gates in the circuits tested. Furthermore, the relation between the percentage of H gates before and after the pre-compilation step is very similar in both circuit types. This relation is expected because:



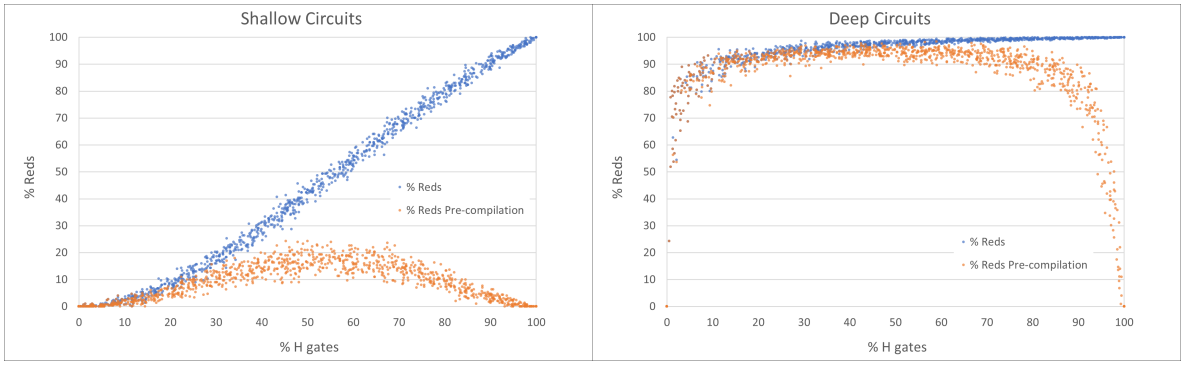
(a) Percentage of H gates after the pre-compilation step for 1010 random shallow circuits ($d = 4$ and $n = 100$).

(b) Percentage of H gates after the pre-compilation step for 1010 random deep circuits ($d = 100$ and $n = 4$).

Figure 29: Percentage of H gates after the pre-compilation step for 1010 random shallow and deep circuits with *h_probability* varying between 0 and 1.

- for a low percentage of H gates ($< 20\%$) in the circuit, not many $HH = I$ substitutions are carried out, so the percentage of H gates after the pre-compilations grows linearly;
- for a percentage of H gates between 20% and 80% the percentage of H gates after the pre-compilation step starts to flatten out, eventually reaching a maximum;
- for a high percentage of H gates ($> 80\%$) more and more adjacent H gates start to appear in the circuits, which leads to a big number of $HH = I$ substitutions, and therefore, to a decrease of the percentage of H gates after the pre-compilation step.

The results concerning the effect of the pre-compilation step on *n_reds* are presented in fig 30. The graphs of figure 30 show that the pre-compilation approach is very effective in



(a) Percentage of *n_reds* before and after the pre-compilation step for 1010 random shallow circuits ($d = 4$ and $n = 100$).

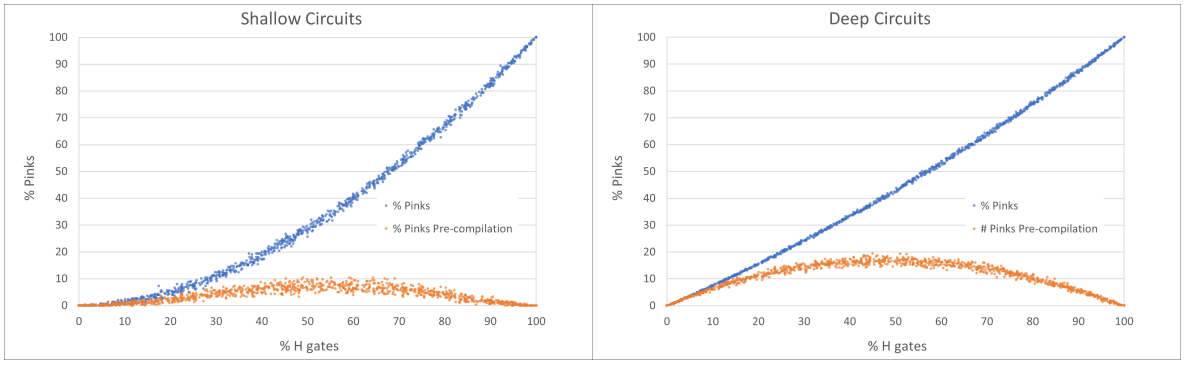
(b) Percentage of *n_reds* before and after the pre-compilation step for 1010 random deep circuits ($d = 100$ and $n = 4$).

Figure 30: Percentage of *n_reds* before and after the pre-compilation step for 1010 random shallow and deep circuits with *h_probability* varying between 0 and 1.

reducing *n_reds* in shallow circuits, for a percentage of H gates greater than 20%, while in

deep circuits this reduction only begins to accentuate for a percentage of H gates greater than 60%. This result is expected because in shallow circuits most of *n_reds* will exist between adjacent H gates, as there is not enough depth for them to exist elsewhere. Figure 30a confirms this as it shows an almost linear relation between the percentage of H gates and *n_reds*. This implies that each $HH = I$ substitution is very likely to eliminate a red colored intermediate state.

Finally, the results comparing *n_pinks* before and after the pre-compilation step are shown in figure 31.



(a) Percentage of *n_pinks* before and after the pre-compilation step for 1010 random shallow circuits ($d = 4$ and $n = 100$).

(b) Percentage of *n_pinks* before and after the pre-compilation step for 1010 random deep circuits ($d = 100$ and $n = 4$).

Figure 31: Percentage of *n_pinks* before and after the pre-compilation step for 1010 random shallow and deep circuits with *h_probability* varying between 0 and 1.

The graphs of figure 31 show that the pre-compilation approach is very effective in reducing *n_pinks* for both shallow circuits and deep circuits. Due to the low percentage of *n_reds* shown in figure 30a in comparison with that in figure 30b, the percentage of *n_pinks* is lower for shallow circuits than it is for deep circuits. However, by comparing figures 30b and 31b, one can note that, for deep circuits, the ratio *n_pinks*/*n_reds* is very low. This shows that the advantage associated with the *PathNB* algorithm described in section 3.2, when compared to the algorithmic method described in section 3.1, is accentuated in deep circuits.

Overall the results presented above provide a quantitative evaluation of the pre-compilation approach and demonstrate its effectiveness in reducing the H count, and therefore, *n_reds* and *n_pinks*. The results presented in figures 30 and 31 imply that applying the pre-compilation step will speed up the simulation process of just about any circuit from the ensemble of circuits used, but the best results, i.e. the highest speedup will be had for shallow circuits with a percentage of H gates greater than around 50%.

4.2.3 *Performance results*

Additional tests were run to compare the performance of both simulators, in which, for each quantum circuit C simulated, a single amplitude $\langle y| C |x\rangle$ was calculated for a single computational basis input state and a single computational basis output state. Each circuit was first simulated using Qiskit's AerSimulator. This simulator supports multiple simulation methods and configurable options for each simulation method. For circuits with a number of qubits lower than 29 the Statevector method was used. For circuits with a number of qubits larger than 29 the Matrix Product State simulation method was used. Then, using the results obtained in those simulations, an output state, $|y\rangle$, for which $\langle y| C |x\rangle$ holds a value greater than zero, was chosen. The input state $|x\rangle$ was chosen to be an all zero n-sized bitstring.

In section 3.2 it was shown that the number of iterations and, consequently, the runtime of *SimulatorNB* grows exponentially with *n_pinks*. To confirm this, square circuits 10×10 ($n = depth = 10$) were simulated, one for each value of *n_pinks* between 6 and 23. The time of each of these simulations was measured and then plotted. These results are presented in figure 32. Figure 32 shows that the time of a simulation using *SimulatorNB* does indeed grow

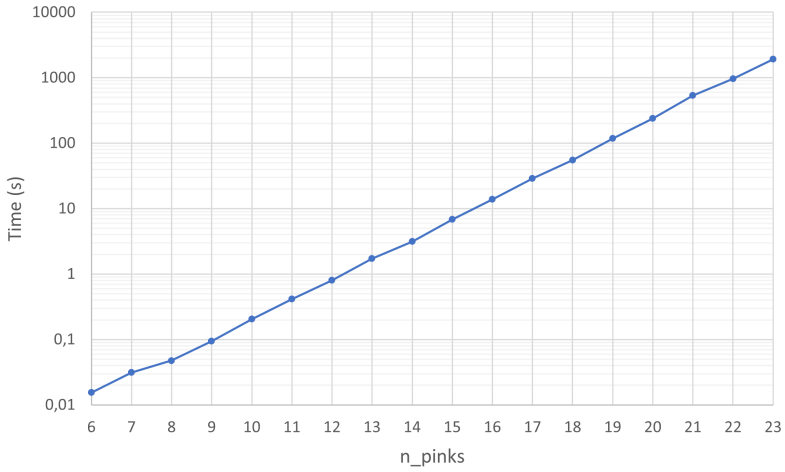


Figure 32: Simulation time using *SimulatorNB* for 10×10 square circuits, with *n_pinks* varying between 6 and 23.

exponentially with *n_pinks*. Moreover, it shows that, in this workstation, for $n\_pinks = 23$, a 10×10 square circuit, takes approximately 33 minutes to complete a simulation. For an experimental testing context where hundreds of circuits are simulated, this value is relatively high. With the results from figures 31 and 32, one can estimate the maximum number *n_pinks* in a circuit, and therefore, its simulation time using *SimulatorNB*, given its size and *h_probability*. The simulation time of *SimulatorRec*, on the other hand, does not depend so

much on the number *n_reds*, but rather, on where do those red colored intermediate states appear in the circuit.

To compare the runtime of simulations using the two simulators, tests were performed on $N \times N$ ($n = d = N$) square circuits, with N ranging from 2 to 10. For each value of N, 50 different circuits were simulated by both simulators with and without the pre-compilation step. Each of these circuits had the parameter *h_probability* set to 0.3. The average over the 50 different circuit simulation times was calculated, for each value of N, and then plotted. Figure 33 presents these results. The graph in figure 33 shows that, for simulations on

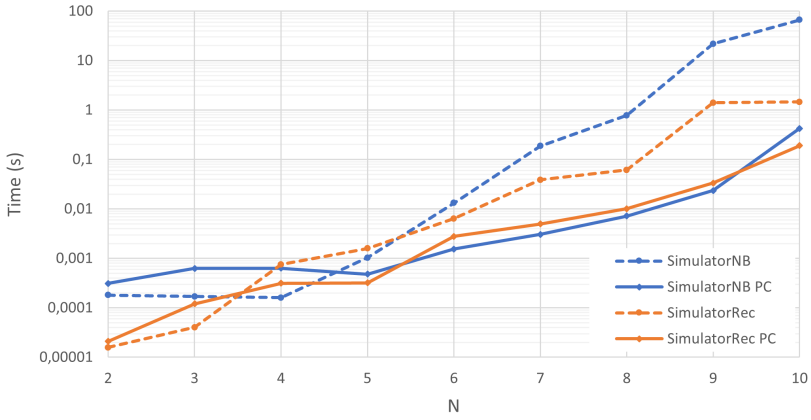


Figure 33: Average simulation time over 50 different square circuits ($n = depth = N$), with *h_probability* set to 0.3, for each value of N.

circuits without pre-compilation, the simulation time of the *SimulatorNB* is the longest for large enough circuits ($N > 5$), and the difference between this and the *SimulatorRec* time tends to increase with increasing circuit size. This is expected as the exponential relation between the simulation time of *SimulatorNB* and *n_pinks* in a circuit makes this simulator more susceptible to an increase in circuit depth. The effectiveness of the pre-compilation step is also evidenced by these results. For $N = 10$ the average simulation time of *SimulatorNB* is, approximately, 160 times shorter for pre-compiled circuits. These results confirm that for large enough circuits, it is always advantageous to incorporate the pre-compilation step. Therefore, for the following tests, the simulations were carried out only for pre-compiled circuits.

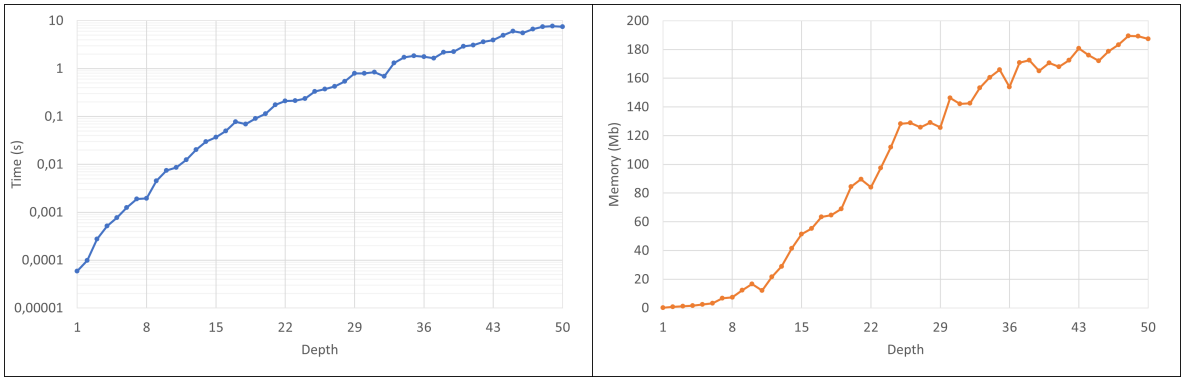
Next, tests were run on shallow circuits with fixed depth and a varying number of qubits, and on deep circuits with a fixed number of qubits and a varying depth, to study the difference in response of these simulators to circuits with the same volume but different form. In the context of these simulator implementations the volume of a circuit, C, on n qubits and with depth d may be defined as:

$$V(C) = nd. \tag{24}$$

In the first of these tests, n was set to 4 and the depth d was set to vary between 1 and 50. For each value of d, 51 different quantum circuits were simulated with *h_probabiliy* varying between 0 and 1. For each simulated circuit, the simulation time and the memory usage were measured and their average values calculated. By inspecting figures 31 and 32, one can estimate the simulation time of a simulation on deep circuits using the *SimulatorNB*. Figure 32 shows that for values of $n\_pinks > 23$ the simulation time is bigger than an hour. Figures 31a and 31b suggest that $10 \leq \%pinks \leq 20$ for the worst case scenario, where $\%Hgates \approx 50$, in pre-compiled circuits. Rearranging eq.(22b) to solve for the depth, d, results in:

$$d = 1 + \frac{100}{\%pinks}\frac{n\_pinks}{n}. \tag{25}$$

Considering a circuit on four qubits, substituting these values into eq.(25) results in $d = 58$ and $d = 29$, for $\%pinks = 10$ and $\%pinks = 20$, respectively. Indeed when running the test described above, many circuits with $n\_pinks > 23$ were generated, for values of d greater than 29, thus resulting in impractical simulation times in the context of the this test. Therefore, it was opted to only run this test with the *SimulatorRec*. It is then concluded that *SimulatorNB* is not suitable for simulating deep circuits. The average simulation time and memory usage of *SimulatorRec* were plotted for each value of d. The results are shown in figure 34.



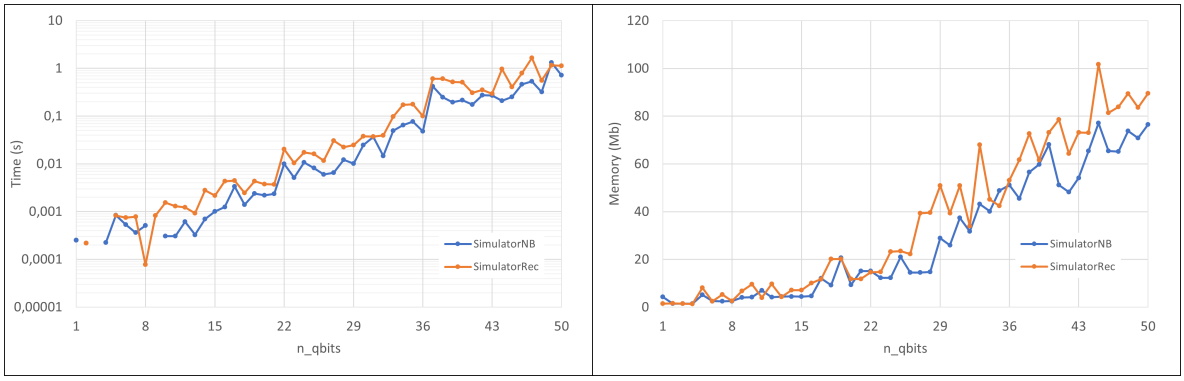
(a) *SimulatorRec*'s average simulation time over 51 different circuits on 4 qubits, with *h_probability* ranging from 0 to 1, for each value of d between 1 and 50.

(b) *SimulatorRec*'s average simulation memory usage over 51 different circuits on 4 qubits, with *h_probability* ranging from 0 to 1, for each value of d between 1 and 50.

Figure 34: *SimulatorRec*'s average simulation time and memory usage over 51 different circuits on 4 qubits, with *h_probability* ranging from 0 to 1, for each value of d between 1 and 50.

Finally, to study how the simulation time and the memory usage evolve with an increasing number of qubits, tests were run on quantum circuits with the number of qubits ranging from 1 to 50. The depth d was set to 4 and, for each value of n, 51 different quantum circuits were simulated with *h_probabiliy* varying between 0 and 1. For each simulated circuit, the simulation time and the memory usage were measured and their average values calculated.

The average simulation time and memory usage of *SimulatorRec* and *SimulatorNB* were plotted for each value of n. The results are shown in figure 35.



(a) Average simulation time over 51 different circuits with $d = 4$ and *h_probability* ranging from 0 to 1, for each value of n between 1 and 50.

(b) Average simulation memory usage over 51 different circuits with $d = 4$ and *h_probability* ranging from 0 to 1, for each value of n between 1 and 50.

Figure 35: Average simulation time and memory usage over 51 different circuits with $d = 4$ and *h_probability* ranging from 0 to 1, for each value of n between 30 and 49.

Figures 34 and 35 present important results regarding the performance of *SimulatorNB* and *SimulatorRec*. While there is certainly room for improvement in terms of speed for these simulations, as discussed in section 5, the experiments conducted demonstrate that these simulators are capable of simulating shallow circuits on a large number of qubits in practical time. Specifically, for $n = 50$ and $d = 4$, both simulators were able to simulate these circuits in an average time of approximately 1 second. Although these simulators have proven effective in simulating shallow circuits, their performance on deep circuits is less satisfactory. This outcome was anticipated and is attributed to the higher percentage of *n_reds* and *n_pinks* observed in deep circuits compared to shallow circuits, as illustrated in figures 30 and 31. The irregular lines plotted in figures 34 and 35 provide evidence that the time and spatial complexities do not depend solely on the circuit dimensions, but also on the circuit structure. Arguably the most significant finding is illustrated in the graph of figure 35b, which demonstrates that the memory usage of both simulators does not increase exponentially with n, in contrast to Schrödinger-type simulators. Additionally, these simulators can simulate circuits with $n = 50$ while utilizing minimal memory (approximately 80 Mb), which is a substantial improvement compared to the memory requirements of a Schrödinger-type simulator, which would need to store all 2^{50} amplitudes. The graph presented in figure 34b indicates that the memory usage of a simulation utilizing *SimulatorRec* also does not exhibit exponential growth with circuit depth. Overall, these findings suggest that these simulators have significant potential for simulating shallow circuits with a large number of qubits ($n > 30$). Nonetheless, there remains considerable room to improve the simulation

time of both simulators. Such enhancements are likely to enable these simulators to simulate circuits with a large number of qubits and higher depths in practical time.

5

CONCLUSION AND FUTURE WORK

Chapter 1 began with a brief historical overview of the origins of quantum computation, starting with the early days of Computer Science. An overview of the importance of the importance of classical quantum simulation is presented later as well as the motivations for studying and improving the existing Feynman path-sum approaches for simulating quantum computers. This introduction is closed with an overview of the following chapters and main contributions.

The goal of chapter 2 was to give an overview of the state of the art of techniques for classical simulation of quantum computers. The Feynman path-sum simulation approach was contextualized as a technique for strong simulation. Other techniques were discussed highlighting the differences between them and providing relevant literature for the development of the work presented in this dissertation.

Chapter 3 was devoted to the study of an optimization method of the Feynman path-sum simulation approach and the development of two simulation algorithms, *PathNB* and *PathRec*, based on this method. These simulation algorithms are the main original contributions of this work and were carefully explained, discussed and visually exemplified in this chapter. The chapter is closed with a brief overview of the benefits of pre-compilation of quantum circuits in general, and specifically in the context of the developed simulators.

Lastly, chapter 4 was dedicated to showcasing experimental results regarding the performance comparison of the simulators implemented with the aforementioned algorithms. The chapter began with the description of the experimental setup, encompassing the set of gates supported by both simulators, the ensemble of circuits on which the experimental tests were made and the implemented pre-compilation rules. Then, the experimental results are presented, starting with the correctness verification of both algorithms. Although the scaling of the tests was limited, the results suggest that both algorithms are correct and the simulation outputs are the same as those in trustworthy simulators. Tests to study the effect of the pre-compilation step on the structure of different circuit types from the ensemble of circuits used, were also carried out. These tests showed how effective a simple pre-compilation step is in reducing the H count, suggesting that an expansion of the pre-compilation rules incorporated in this step would result in a considerable speedup of the simulations. Finally,

tests concerning the performance comparison between both simulators were carried out, focusing on the simulation time and memory usage. These results present evidence of the potential of these simulators. They were able to simulate shallow circuits with $n = 50$ and $d = 4$, where a Schrödinger-type simulator would consume excessive amounts of memory, in practical time (approximately 1 second per circuit) and using low memory. The time complexity was proven to depend mainly on the circuit structure, rather than the circuit volume. The memory complexity also depends on the circuit structure but appears to grow linearly with n and d. Tests for deep circuits were not as satisfactory, as expected, but there is still space for improvement, especially regarding the simulation time. This indicates that there is potential to improve the efficiency of these simulators in simulations on these types of circuits.

Based on what was achieved in this dissertation, in order to further improve the efficiency and expand the capabilities of the current implementations of *SimulatorNB* and *SimulatorRec*, several potential paths for future work are suggested. These include:

- Code optimization: Fine-tuning the simulation algorithm to optimize for the specific gate set being used. This could involve identifying specific types of circuits or gate combinations that are particularly computationally expensive, and developing targeted optimizations to reduce the overhead of these operations. During the calculation of amplitudes for two different paths, it is possible that many computations may be repeated. To improve efficiency, the algorithms can be optimized to identify and avoid redundant calculations. Additionally, there can be an investigation of whether or not the recursive algorithm used in *SimulatorRec* should be re-implemented as an iterative algorithm. Trade-offs between efficiency, readability, and maintainability should be considered in this potential optimization.

- Faster programming language: Re-implementing the simulators in a faster programming language, such as C++. While the existing implementation in Python is flexible and easy-to-use, it may not be ideal for computationally intensive tasks involving large circuits.

- Larger gate set: Expanding the gate set used by the simulator to include additional types of gates beyond the current set would enable simulation of more complex circuits. It could also facilitate the incorporation of new gate identities in a pre-compilation step.

- More pre-compilation identities: As shown throughout 4.2, the power of a pre-compilation step is immense, even when using only a single identity. By incorporating more identities that decrease the number of H gates in a circuit, such as those presented in figure 25, or identities that decrease the number of other branching gates in a circuit,

in the case such gates are added to the gate set, a great speedup in the simulation times of both simulators is expected.

- Parallel and/or distributed computation: Explore the use of parallel and distributed computation to further speed up simulations. Feynman path-sum based algorithms are inherently parallelizable. This is because these algorithms involve performing a large number of similar calculations on different paths, which makes it suitable to split the simulation workload across multiple processors or computers. Parallelization can significantly speed up the computation of simulations using both implemented simulators, especially for circuits with a large number of qubits and/or large depth.

- Hybrid Schrödinger/Feynman approach: Finally, future work could be made to study, implement and test simulators that combine *SimulatorNB* and *SimulatorRec* with a Schrödinger type simulator. Aaronson and Chen (2016) described an hybrid Schrödinger/Feynman simulator and showed that, by interpolating the two approaches, one achieves a tradeoff between the memory usage and the simulation runtime.

BIBLIOGRAPHY

Aaronson, S. and Chen, L. (2016), 'Complexity-theoretic foundations of quantum supremacy experiments'. arXiv:1612.05903.
URL: *https://arxiv.org/abs/1612.05903*

Aaronson, S. and Gottesman, D. (2004), 'Improved simulation of stabilizer circuits', *Phys. Rev. A* **70**, 052328.
URL: *https://link.aps.org/doi/10.1103/PhysRevA.70.052328*

Abdessaied, N., Soeken, M. and Drechsler, R. (2014), Quantum circuit optimization by hadamard gate reduction, *in* S. Yamashita and S.-i. Minato, eds, 'Reversible Computation', Springer International Publishing, Cham, pp. 149–162.

Arute, F., Arya, K., Babbush, R., Bacon, D., Bardin, J., Barends, R., Biswas, R., Boixo, S., Brandao, F., Buell, D., Burkett, B., Chen, Y., Chen, Z., Chiaro, B., Collins, R., Courtney, W., Dunsworth, A., Farhi, E., Foxen, B. and Martinis, J. (2019), 'Quantum supremacy using a programmable superconducting processor', *Nature* **574**, 505–510.

Backens, M. (2014), 'The zx-calculus is complete for stabilizer quantum mechanics', *New Journal of Physics* **16**(093021).
URL: *https://dx.doi.org/10.1088/1367-2630/16/9/093021*

Benioff, P. (1980), 'The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines', *Journal of Statistical Physics* **22**(5), 563–591.

Bernstein, E. S. and Vazirani, U. V. (1993), 'Quantum complexity theory', *Proceedings of the twenty-fifth annual ACM symposium on Theory of Computing* .

Boixo, S., Isakov, S. V., Smelyanskiy, V. N. and Neven, H. (2017), 'Simulation of low-depth quantum circuits as complex undirected graphical models'. arXiv:1712.05384.
URL: *https://arxiv.org/abs/1712.05384*

Bravyi, S., Browne, D., Calpin, P., Campbell, E., Gosset, D. and Howard, M. (2019), 'Simulation of quantum circuits by low-rank stabilizer decompositions', *Quantum* **3**, 181.
URL: *https://doi.org/10.22331%2Fq-2019-09-02-181*

Brod, D. J. (2014), 'The computational power of non-interacting particles'. arXiv:1412.7637.

Chuang, I. L., Gershenfeld, N. A. and Kubinec, M. (1998), 'Experimental implementation of fast quantum searching', *Physical Review Letters* **80**, 3408–3411.

Coecke, B. and Duncan, R. (2008), Interacting quantum observables, *in* L. Aceto, I. Damgaard, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir and I. Walukiewicz, eds, 'Automata, Languages and Programming', Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 298–310.

Corboz, P., Jordan, J. and Guifré Vidal (2010), 'Simulation of fermionic lattice models in two dimensions with projected entangled-pair states: Next-nearest neighbor hamiltonians', *Physical Review B* **82**(245119).
URL: *https://doi.org/10.1103%2Fphysrevb.82.245119*

Dang, A., Hill, C. D. and Hollenberg, L. C. L. (2019), 'Optimising Matrix Product State Simulations of Shor's Algorithm', *Quantum* **3**, 116.
URL: *https://doi.org/10.22331/q-2019-01-25-116*

De Raedt, H., Jin, F., Willsch, D., Willsch, M., Yoshioka, N., Ito, N., Yuan, S. and Michielsen, K. (2019), 'Massively parallel quantum computer simulator, eleven years later', *Computer Physics Communications* **237**, 47–61.
URL: *https://www.sciencedirect.com/science/article/pii/S0010465518303977*

den Nest, M. V. (2009), 'Classical simulation of quantum computation, the gottesman-knill theorem, and slightly beyond'. arXiv:0811.0898.

Deutsch, D. (1985), 'Quantum theory, the church–turing principle and the universal quantum computer', *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* **400**, 97 – 117.

Deutsch, D. and Jozsa, R. (1992), 'Rapid solution of problems by quantum computation', *Proceedings: Mathematical and Physical Sciences* **439**, 553–558.

Duan, R., Wu, H. and Zhou, R. (2022), 'Faster matrix multiplication via asymmetric hashing'. arXiv:2210.10173.
URL: *https://arxiv.org/abs/2210.10173*

Duncan, R., Kissinger, A., Perdrix, S. and van de Wetering, J. (2020), 'Graph-theoretic simplification of quantum circuits with the ZX-calculus', *Quantum* **4**, 279.
URL: *https://doi.org/10.22331%2Fq-2020-06-04-279*

Fatima, A. and Markov, I. L. (2020), 'Faster schrödinger-style simulation of quantum circuits'. arXiv:2008.00216.
URL: *https://arxiv.org/abs/2008.00216*

Feynman, R. (1948), 'Space-Time Approach to Non-Relativistic Quantum Mechanics', *Reviews of Modern Physics* **20**(2), 367–387.

Feynman, R. (1959), 'There's plenty of room at the bottom', *Feynman and Computation*. 63-76.

Feynman, R. (1982), 'Simulating physics with computers', *International Journal of Theoretical Physics* **21**, 467–488.

Gottesman, D. (1998), 'The heisenberg representation of quantum computers'. arXiv:quant-ph/9807006.
URL: *https://arxiv.org/abs/quant-ph/9807006*

Grover, L. K. (n.d.), 'A fast quantum mechanical algorithm for database search', *STOC '96: Proceedings of the twenty-eighth annual ACM symposium on Theory of Computing* pp. 212–219.

Guo, C., Liu, Y., Xiong, M., Xue, S., Fu, X., Huang, A., Qiang, X., Xu, P., Liu, J., Zheng, S., Huang, H.-L., Deng, M., Poletti, D., Bao, W.-S. and Wu, J. (2019), 'General-purpose quantum circuit simulator with projected entangled-pair states and the quantum supremacy frontier', *Physical Review Letters* **123**(190501).
URL: *https://doi.org/10.1103%2Fphysrevlett.123.190501*

Heyfron, L. E. and Campbell, E. T. (2018), 'An efficient quantum compiler that reduces t count', *Quantum Science and Technology* **4**(015004).
URL: *https://dx.doi.org/10.1088/2058-9565/aad604*

Huang, Y. and Love, P. (2021), 'Feynman-path-type simulation using stabilizer projector decomposition of unitaries', *Physical Review A* **103**(022428).
URL: *https://doi.org/10.1103%2Fphysreva.103.022428*

Jeandel, E., Perdrix, S. and Vilmart, R. (2018), 'A complete axiomatisation of the zx-calculus for clifford+t quantum mechanics'. arXiv:1705.11151.

Jozsa, R., Kraus, B., Miyake, A. and Watrous, J. (2009), 'Matchgate and space-bounded quantum computations are equivalent', *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* **466**(2115), 809–830.
URL: *https://doi.org/10.1098%2Frspa.2009.0433*

Kissinger, A. and van de Wetering, J. (2022), 'Simulating quantum circuits with ZX-calculus reduced stabiliser decompositions', *Quantum Science and Technology* **7**(044001).
URL: *https://doi.org/10.1088%2F2058-9565%2Fac5d20*

Luchnikov, I. A., Berezutskii, A. V. and Fedorov, A. K. (2021), 'Simulating quantum circuits using the multi-scale entanglement renormalization ansatz'. arXiv:2112.14046.
URL: *https://arxiv.org/abs/2112.14046*

Markov, I. L., Fatima, A., Isakov, S. V. and Boixo, S. (2018), 'Quantum supremacy is both closer and farther than it appears'. arXiv:1807.10749.
URL: *https://arxiv.org/abs/1807.10749*

Markov, I. L. and Shi, Y. (2008), 'Simulating quantum computation by contracting tensor networks', *SIAM Journal on Computing* **38**(3), 963–981.
URL: *https://doi.org/10.1137%2F050644756*

Mooney, G. J., Hill, C. D. and Hollenberg, L. C. L. (2021), 'Cost-optimal single-qubit gate synthesis in the Clifford hierarchy', *Quantum* **5**, 396.
URL: *https://doi.org/10.22331/q-2021-02-15-396*

Moore, G. E. (1965), 'Cramming more components onto integrated circuits', *Electronics* **38**(8), 114–117.

Pan, F. and Zhang, P. (2021), 'Simulating the sycamore quantum supremacy circuits'. arXiv:2103.03074.
URL: *https://arxiv.org/abs/2103.03074*

Pednault, E., Gunnels, J. A., Nannicini, G., Horesh, L. and Wisnieff, R. (2019), 'Leveraging secondary storage to simulate deep 54-qubit sycamore circuits'. arXiv:1910.09534.
URL: *https://arxiv.org/abs/1910.09534*

Preskill, J. (2012), 'Quantum computing and the entanglement frontier'. arXiv:1203.5813.

Preskill, J. (2018), 'Quantum Computing in the NISQ era and beyond'. arXiv:1801.00862.

Shor, P. W. (1994), 'Algorithms for quantum computation: discrete logarithms and factoring', *Proceedings 35th Annual Symposium on Foundations of Computer Science* pp. 124–134.

Terhal, B. M. and DiVincenzo, D. P. (2002), 'Classical simulation of noninteracting-fermion quantum circuits', *Physical Review A* **65**(032325).
URL: *https://doi.org/10.1103%2Fphysreva.65.032325*

Turing, A. M. (1937), 'On computable numbers, with an application to the entscheidungsproblem', *Proceedings of the London Mathematical Society* **s2-42**(1), 230–265.
URL: *https://londmathsoc.onlinelibrary.wiley.com/doi/abs/10.1112/plms/s2-42.1.230*

Valiant, L. (2002), 'Quantum circuits that can be simulated classically in polynomial time', *SIAM J. Comput.* **31**, 1229–1254.

van de Wetering, J. (2020), 'Zx-calculus for the working quantum computer scientist'. arXiv:2012.13966.

Vidal, G. (2003), 'Efficient classical simulation of slightly entangled quantum computations', *Physical Review Letters* **91**(147902).
URL: *https://doi.org/10.1103%2Fphysrevlett.91.147902*