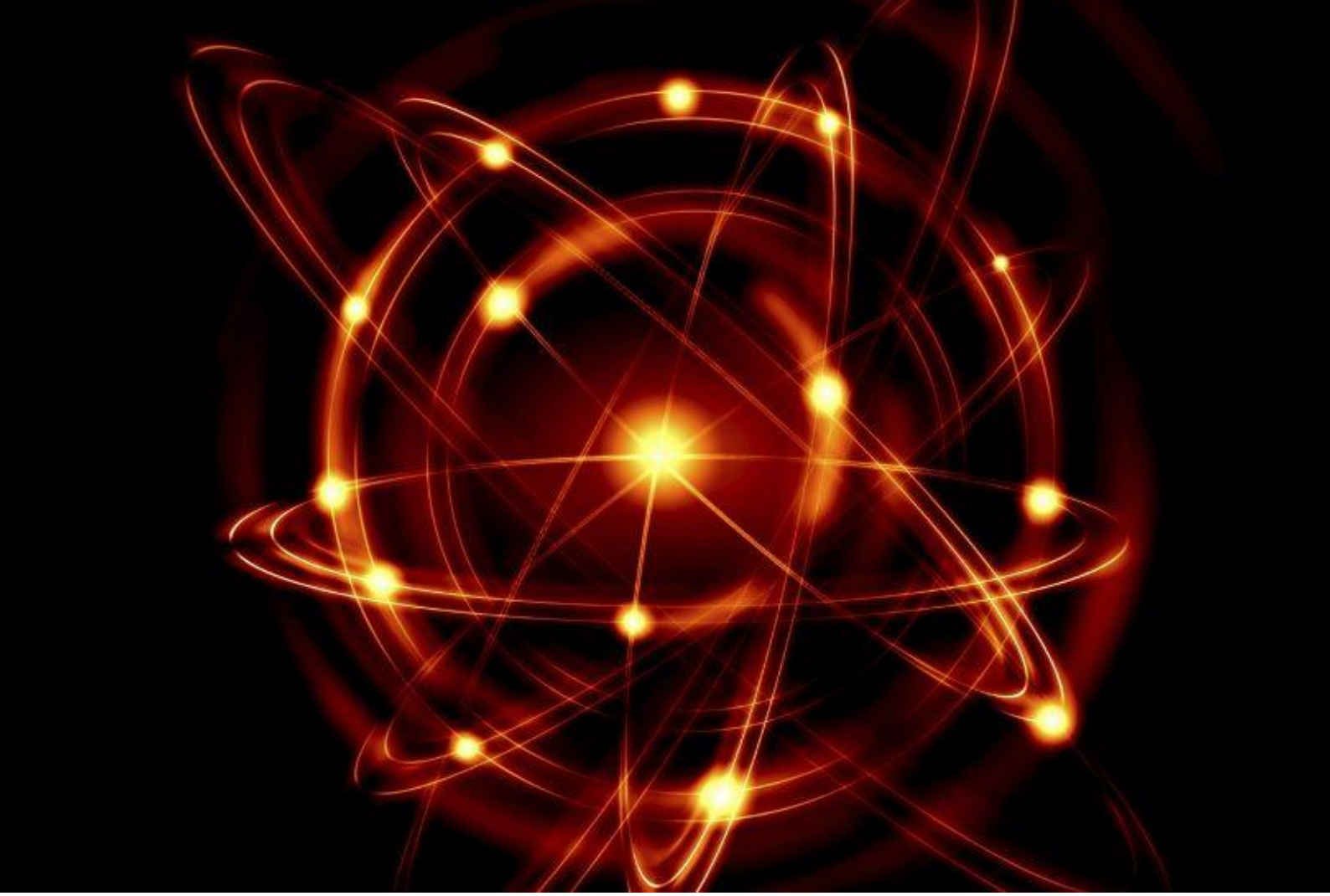


Resource Optimal Executable Quantum Circuit Generation Using Approximate Computing

By

Smaran Adarsh

Resource Optimal Executable Quantum Circuit Generation Using Approximate Computing

by

Smaran Adarsh

to obtain the degree of,
Master of Science in Computer Engineering,
at the Delft University of Technology,
to be defended publicly on Tuesday July 6, 2021 at 15:00.

Student number:	5124948	
Project duration:	September 1, 2020 – June 30, 2021	
Thesis committee:	Dr. rer. nat. Matthias Möller,	TU Delft, Daily supervisor
	Dr. ir. Zaid Al-Ars,	TU Delft, Responsible supervisor
	Ir. Aritra Sarkar,	TU Delft

An electronic version of this thesis is available at http://repository.tudelft.nl/.

Abstract

Quantum computing is an emerging technology that combines the principles of both computer science and quantum mechanics to solve computationally challenging problems significantly faster than the current classical computers. In this thesis, a proof of concept to generate hardware-executable quantum circuits for Noisy Intermediate-Scale Quantum (NISQ) devices that follow the paradigm of approximate computing is presented.

We adopt a multi-level optimization approach and consider the placement of native quantum gates that can be physically implemented on one of the existing quantum systems (`IBM`, `Rigetti`) as a *circuit topology optimization* problem and the adjustment of parameters of these native quantum gates as a *continuous design optimization* problem. The outer topology problem is solved with the aid of a genetic algorithm, whereas the gradient-descent method is used for the inner design optimization.

Our approach starts from the reference circuit and transforms it into an executable circuit with tuneable parameters by replacing the high-level quantum operations with a set of approximate decompositions, e.g., single qubit rotation quantum gates or a combination of single qubit rotation gates (matrix multiplication). Later, the inner design optimization ensures that the circuit created is tuned to be executed to achieve an expectation value close to that of the reference circuit. This is complemented by *compiler-based optimizations* to further reduce or aggregate the quantum gates of the optimized circuit. This three-step pipeline is embedded into an outer genetic algorithm framework that inspects many different circuit designs (the approximate-decomposition replacements are by no means unique) and returns a set of unique approximate quantum circuits that can be executed on hardware.

We have tested our circuit generation approach for superconducting quantum systems such as `IBM` and `Rigetti` hardware for many different benchmark circuits such as *Quantum Fourier Transform*, *Toffoli*, *quantum adder*, *multi-controlled operations* and three oracle-based algorithms, namely, *Grover's search* algorithm, *Deutsch* and *Bernstein-Vazirani* algorithm. In nearly all cases, our approach outperforms the quantum compiler frameworks by `IBM` and `Rigetti` even on their highest optimization level and produces significantly smaller circuits, up to 2x reduction in the number of gates. In addition, the circuits generated using the proposed approach provide better (a) performance (on average, improved by 24.25%) and (b) reliability (on average, improved by 23.50%) compared to the `IBM` generated circuits when executed on the `ibmq_5_yorktown` physical quantum device.

Acknowledgments

I would like to thank Matthias Möller for his constant encouragement, able guidance and for providing me with this opportunity to build and strengthen my knowledge in the field of quantum computing. I would also like to thank Zaid Al-Ars for his valuable support and feedback, which immensely helped me to improve and shape my thesis.

I would like to acknowledge the entire quantum computing group for their assistance during my thesis work. I would also like to express my gratitude towards my parents and friends for their unconditional support, patience and encouragement. Without them, this would not have been possible.

Smaran Adarsh
Delft, The Netherlands
July 6, 2021

Contents

Abstract **iii**

Acknowledgments **v**

1 Introduction **1**
1.1 Context . . . 1
1.2 Motivation . . . 3
1.3 Aim and scope . . . 4
1.4 Thesis layout . . . 5

2 Literature survey **7**
2.1 Quantum computing . . . 7
2.1.1 Qubit representation . . . 11
2.1.2 Physical implementation . . . 13
2.1.3 Quantum gates . . . 15
2.1.4 Quantum algorithms . . . 22
2.2 Classical optimization . . . 24
2.3 Expectation value . . . 25
2.4 Approximate computing . . . 26
2.5 Genetic algorithm . . . 27
2.6 Training parameterized quantum circuits . . . 28

3 Generation of executable approximate circuits **33**
3.1 Rule-based approximate gate substitution . . . 35
3.2 Continuous parameter optimization . . . 39
3.3 Compiler-based optimization . . . 42
3.4 Approximate circuit generation for `IBM` quantum device . . . 49

4 Genetic algorithm for circuit generation **51**
4.1 Introduction to genetic algorithm . . . 51
4.2 Optimal circuit using the genetic algorithm . . . 53

5 Experimental evaluation and discussion **57**
5.1 Resource reduction . . . 57
5.1.1 Quantum circuits without oracles . . . 57
5.1.2 Quantum circuits with oracles . . . 59
5.2 Noise reduction . . . 60
5.3 Execution time reduction . . . 66

6 Limitations **67**

7 Conclusion **73**
7.1 Conclusion and findings . . . 73
7.2 Answers to the research questions . . . 74

8 Future work **77**

Appendices **89**

A Case Study: CNOT Gate Implementation **91**

List of Figures

1.1 Quantum circuit to create the first Bell state [1]. . . . 1

2.1 Preparation of the first bell state. . . . 8
2.2 Quantum teleportation circuit [2]. . . . 9
2.3 Quantum superdense coding circuit [2]. . . . 9
2.4 Quantum entanglement swap circuit [2]. . . . 10
2.5 Single qubit Bloch sphere representation [2]. . . . 12
2.6 The action of the *Pauli-X* gate on the $|0\rangle$ state. . . . 12
2.7 The action of the *hadamard* gate on the $|1\rangle$ state. . . . 12
2.8 Electoral oscillators [3]. . . . 13
2.9 *Not* gate and the corresponding truth table [4]. . . . 14
2.10 *AND* gate and the corresponding truth table [4]. . . . 14
2.11 X gate. . . . 15
2.12 Y gate. . . . 16
2.13 Z gate. . . . 16
2.14 *Hadamard* gate. . . . 16
2.15 R_x gate. . . . 17
2.16 R_y gate. . . . 17
2.17 R_z gate. . . . 18
2.18 SX gate. . . . 18
2.19 S gate. . . . 18
2.20 T gate. . . . 19
2.21 *CNOT* gate. . . . 19
2.22 *CZ* gate. . . . 20
2.23 *Toffoli* gate. . . . 20
2.24 *Deustch* algorithm. . . . 23
2.25 2 qubit *Grover's* algorithm. . . . 23
2.26 Optimization of rotation angles [5]. . . . 25
2.27 General representation of born machine [6]. . . . 30

3.1 Flow chart of the multi-level optimization algorithm. The reiteration from circuit optimizer to *rule-based approximate gate substitution* (dashed arrow) is not yet included in our proof-of-concept implementation but supported conceptually. 33
3.2 `Rigetti` compiler generated *CNOT* circuit. . . . 34
3.3 Approximate *CNOT* circuit generated using the proposed algorithm. . . . 34
3.4 The action of the *hadamard* gate on the $|0\rangle$ state. . . . 36
3.5 The action of the *hadamard* gate on the $|+\rangle$ state. . . . 36
3.6 Bell state preparation circuit using *hadamard* gate. . . . 36
3.7 Bell state preparation circuit using R_y gate. . . . 37
3.8 Exact *CNOT* gate using *hadamards*. . . . 37
3.9 Approximate *CNOT* gate. . . . 37
3.10 *Rule-based approximate gate substitution* for *CNOT* circuit. . . . 38
3.11 Approximate *CNOT* circuit with user-defined threshold = 0. . . . 40

3.12 Approximate *Toffoli* circuit (for a `IBM` hardware) with user-defined threshold = 0.0857. . . . 40
3.13 Reference *Quantum Fourier Transform* circuit. . . . 41
3.14 Approximate *Quantum Fourier Transform* circuit that works for any given input state. . . . 41
3.15 Approximate *Quantum Fourier Transform* circuit that works only for the computational basis states. . . . 41
3.16 *CNOT* after *rule-based approximate substitution.* . . . 42
3.17 Unitary gate negation. . . . 43
3.18 Negation of similar rotation gates. . . . 44
3.19 Aggregation of similar rotation gates. . . . 44
3.20 Gate count reduction using the commutativity property of the quantum gates. 45
3.21 Specific quantum gate identities. . . . 45
3.22 Generic quantum identities . . . 46
3.23 Negation of unaltered parameterized quantum gates. . . . 47
3.24 Approximate `Rigetti` executable *CNOT* circuit generated from the proposed algorithm. . . . 48
3.25 Approximate `IBM` executable *BM1* circuit generated from the proposed algorithm. 50

4.1 Flow chart of our genetic algorithm. . . . 52
4.2 2 qubit *Grover's* algorithm. . . . 54
4.3 Approximate 2 qubit *Grover's* algorithm generated to be performed on a `IBM` quantum device. . . . 56
4.4 Approximate 2 qubit *Grover's* algorithm generated to be performed on a `Rigetti` quantum device. . . . 56

5.1 `ibmq_5_yorktown` qubit architecture [7]. . . . 62
5.2 Effect of real noise on the 2 qubit *Grover's* circuit. . . . 62
5.3 Effect of real noise on the *Toffoli* circuit. . . . 63
5.4 Effect of real noise on the 2 qubit *quantum adder* circuit. . . . 63
5.5 Effect of real noise on the *quantum half adder* circuit. . . . 64
5.6 Effect of real noise on the *inverse peres* circuit. . . . 64
5.7 Effect of real noise on the *quantum nand* circuit. . . . 65
5.8 Effect of real noise on the *fredkin* circuit. . . . 65

6.1 Reference *Quantum Fourier Transform* circuit. . . . 68
6.2 `Rigetti` compiled *Quantum Fourier Transform* circuit. . . . 68
6.3 Approximate *Quantum Fourier Transform* circuit generated using the propose algorithm. . . . 68
6.4 Reference circuit A. . . . 69
6.5 Approximate circuit A^* generated using the proposed algorithm. . . . 69
6.6 Reference circuit $(A + B)$. . . . 69
6.7 Approximate circuit $(A^* + B^*)$ using the proposed algorithm. . . . 69
6.8 Approximate circuit $(A + B)^*$ generated from the reference circuit $(A + B)$. . 70

A.1 The effect of real noise as the number of *CNOT* operations increase. . . . 91

List of Tables

1.1 Native gates of different quantum systems used in our study. The quantum gates mentioned in the table are briefly described in Section 2.1.3 [8]. 2

5.1 Resource evaluation of our approach for quantum benchmark circuits generated for the `IBM` platform. 58
5.2 Resources evaluation of our approach for quantum benchmark circuits generated for the `Rigetti` platform. 59
5.3 Resource evaluation of our approach for oracle-based quantum benchmark circuits generated for the `IBM` platform. 60
5.4 Resource evaluation of our approach for oracle-based quantum benchmark circuits generated for the `Rigetti` platform. 60
5.5 Average error rate and the execution time associated with all the `ibmq_5_yorktown` supported native gates. In the case of the CNOT(a,b) operation, a is the control qubit whereas b is the target qubit. 61
5.6 Number of physical CNOT operations required to implement the quantum benchmark circuit on the the `ibmq_5_yorktown` device. 61
5.7 Execution time comparison of our approach for quantum benchmark circuits generated for the `IBM` platform. 66

1 Introduction

1.1 Context

Quantum computing is an emerging computing technology that has the potential to radically change the way we will be solving computational problems in the future. While classical computers deal with classical bits (0's and 1's) to perform computation, quantum computers perform computation using qubits, which can perform computation by being in state 0 and 1 simultaneously (superposition) [9].

In the '90s, quantum computers were considered to be hypothetical machines that were based on the general principles of quantum mechanics (superposition and entanglement) and were expected to perform computation more efficiently when compared to the state of the art classical computers [10].

After decades of theoretical research, the last years have brought the prototype quantum systems [11]. However, the noisiness, the small number of available qubits and the constraint of low quantum volume on the Noisy Intermediate-Scale Quantum (NISQ) [12] devices are often considered a severe drawback that hinders their applicability to real-world problems [13]. At the same time, it also provides motivation to rethink traditional approaches that have been established over decades of classical computing but might be unhandy in the quantum era.

Though fundamentally different from other computing technologies, quantum computers likewise require a full software stack [8, 14]. Over the last few years, prototypes of full-stack quantum computers have been released by `IBM` and others enabling researchers and practitioners to explore this fascinating emerging technology that has the potential to revolutionize the way of computing in the future. It has also enabled application developers to express algorithms in a human-readable programming language and transform them into executable instructions through a quantum compiler toolchain [15].

In classical computing, compiler toolchains fulfill the nontrivial task of transforming human-readable source code into an executable machine code thereby applying all types of optimizations and sanity checks. The development of quantum compilers is, as the entire quantum technology, still in its infancy.

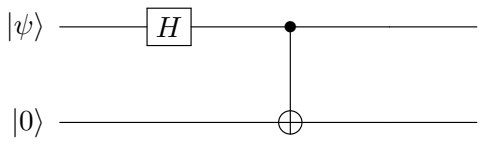


Figure 1.1: Quantum circuit to create the first Bell state [1].

Most of today's quantum-compiler approaches follow the traditional concept of transforming an existing reference circuit consisting of a set of quantum operations (gates) into an executable circuit by a rule-based gate (-group) substitution where the high-level quantum operations are substituted by the equivalent set of native quantum gates or machine learning-

based creation of a surrogate circuit that the quantum device supports [16]. In both cases, the goal is to create a substitute circuit that is close to the reference one, i.e., both circuits map the same input state to the same output state, thereby using only the hardware's native gate set and respecting the sparse qubit connectivity [17].

Table 1.1: Native gates of different quantum systems used in our study. The quantum gates mentioned in the table are briefly described in Section 2.1.3 [8].

`IBM`	`Rigetti`
X, $R_x(\frac{\pi}{2})$, $R_z(\lambda)$, CR[1]	$R_x(\pm k\frac{\pi}{2})$, $k \in \mathbb{N}$, $R_z(\lambda)$, CZ, XY[2]

A common approach in programming gate-based quantum computers is to adopt one of the many quantum assembly languages [7, 18, 5] and implement *quantum algorithms* by applying single- and multi-qubit quantum gates to the quantum register $|\psi\rangle$ (Figure 1.1). The vertical and horizontal dimensions of a quantum circuit are known as its *depth* and *width*, respectively. We will refer to circuits of this type as *reference circuits* as they serve as abstract models that cannot be executed directly on physical quantum devices.

This is where quantum compiler frameworks come into play. They decompose [19] all software-visible gates into native gates (Table 1.1), schedule these native gates based on their co-dependence with each other, map between software and physical qubits, insert additional swap operations to enable multi-qubit operations between non-connected qubits, optimize the circuit to reduce its gate count and the overall size of the circuit, and generate electric signals (called control pulses) to implement the circuit on a specific physical quantum device [20].

These compilers make sure that the circuit generated does not alter the mapping between the basis states when compared to the reference circuit. However, creating this exact mapping between the input and the output states comes with the cost of additional quantum gates which in turn increases the circuit size. This is where an interesting observation can be made. From the practical implementation point of view, exact mapping is not required as long as we can read out or measure the correct output state for any given input state with the probability same as what we would expect from the exact mapping.

What if we can generate circuits that instead of exact mapping produces an approximate mapping between the basis states but still results in the same outcome as the reference circuit when measured on a physical device thereby using only the native gates which are supported on the given quantum system and overall results in a hardware-executable circuit with size significantly less compared to the circuit generated by the traditional quantum compilers? As a result of this, we can improve the circuit's

- reliability (reduce the circuit's susceptibility to noise).
- the execution time on the real hardware.

This idea captures the main motivation behind this thesis in which we investigate whether this possible or not. If it is possible, what is the general procedure that can be followed to implement this idea?

[1]The cross resonance (CR) gate is considered to be the *CNOT* quantum gate according to the implementation on the `IBM` hardware via Qiskit[7].

[2]The paramaterized ISWAP (XY) gate has been introduced recently with PyQuil [18] version 2.22.0 and is not yet considered in our multi-level circuit optimization approach.

1.2 Motivation

On an abstract level, quantum computers, similar to their classical counterpart, are constructed using basic quantum gates. Analogous to the *MOSFETs* used to implement the classical logic gates [21], quantum computers are constructed using superconducting electronics where single or multi-qubit operations that are supported on specific hardware can be performed by generating microwave pulses to either excite the state of the electron or bring them back to the ground state. Each hardware supports a set of native gates that are used to perform any form of quantum computation and they are described in Table 1.1.

In classical computing, computations can be performed by constructing the circuits using available logic gates. The performance of these circuits is highly dependent on

- the size of the circuit (total number of logic gates used).
- depth of the circuit (critical path of the circuit).

Moreover, the size of the circuit determines whether

- the circuit is cost-effective.
- the circuit is energy efficient.

Similar to the above observation, the performance of the quantum circuit can be characterized and evaluated based on the total number of operations that have been used to implement the circuit. However, a major difference between the classical and the quantum computing world is in terms of the corresponding data (information) holders, i.e., bits and the qubits, respectively. Unlike their classical counterpart, the concept of decoherence [22] is introduced in the quantum world.

The qubits are sensitive, fragile and have a short coherence time. This makes the qubits and the quantum world to be more prone to errors. The errors can be visualized and modeled as

- bitflip [23],
- phase flip [24],
- amplitude damping [25],
- phase damping [26],
- depolarizing [27],
- single qubit or combination of single qubit errors.

In a quantum circuit, as the qubit passes from one quantum gate to another, it is uncertain that the quantum state will remain unaltered (unaffected) by noise or other environmental factors. Moreover, due to the fragility of the qubit, the strength of the qubit to retain the information is weakened with every gate it passes through. As a result, the size (total number of gates) of the quantum circuit plays a very important role in the NISQ era. With this notion, many compilers with various optimization levels are introduced to reduce the overall count of the operations with the hope of having a less error-prone physically implementable circuit. The lesser the number of quantum gates that are used to execute the circuit, the

lower is the probability that the corresponding quantum state (qubits) is prone to noise and other environmental factors. In addition, assuming that each quantum operation is physically implemented as a cost factor, quantum circuits that require fewer quantum gates are more cost-effective as well.

Overall, currently in the quantum computing world, the goal is to construct quantum circuits with the least number of possible resources (quantum gates) so that the resulting quantum circuits are

- less prone to errors,
- cost-effective,
- performance-wise executed efficiently.

This also affects the quantum volume of any given circuit [28]. Quantum volume is one of the most important metrics in the quantum computing world that was introduced by `IBM` which is used to measure the performance of today's quantum computers. Quantum volume is calculated based on various factors such as the number of qubits required, total number of gates implemented, qubit and measurement error rates. Hence, a quantum circuit executed with the least number of resources (quantum operations) results in having a low quantum volume number which in turn means that the circuit can be executed with high performance.

The above-addressed issues motivate our aim to generate hardware-executable approximate quantum circuits for any given reference circuit by using the least number of resources. As a result, there is a reduction in the size, noise susceptibility and execution time of these circuits. Although the circuit generated is approximate for the reference circuit, our approach promises to produce the measurement outcomes with the same probability as the considered reference circuit. Our approach has been performed using both the `IBM` and `Rigetti` supported native gates as shown in Table 1.1.

1.3 Aim and scope

This thesis aims towards providing a new way of generating hardware-executable quantum circuits for the NISQ devices with the assistance of approximate computing. The approach to generate hardware-executable quantum circuits is inspired by topology design optimization as it is applied in many aerospace, mechanical and civil engineering applications [29]. Common engineering practice is to investigate multiple topology configurations and optimize their tunable parameters individually before selecting the overall winning design. In particular, our approach is tested for the superconducting quantum systems of `IBM` and `Rigetti` and is expected to work for other platforms as well.

The concept of approximate computing [30] is introduced by creating executable circuits that mimic the expectation-value landscape of the reference circuit most accurately instead of capturing the exact mapping between input and output states. We consider the former more important than the latter as states cannot be read out from real quantum hardware anyway and the only accessible data are the measurement outcomes.

Our computational approach consists of four main components. First, all software-visible gates of the reference circuit are replaced by approximate decompositions (combination of $R_z(\theta)$ and $R_x(\theta)$), thereby rotation angles are treated as design parameters that are optimized by a gradient-descent optimizer in the second step. Optimization is performed for a

cost function that measures the discrepancy between the expectation-value landscape of the reference and the executable circuit. The converged circuit is post-processed by a compiler-based optimization sweep to eliminate canceling gate groups, fuse consecutive rotations of the same type, and aggregate gate groups using commutativity rules to reduce the circuit's depth and its gate count. This three-step pipeline is embedded into an outer genetic algorithm framework that inspects many different circuit designs (the approximate-decomposition replacements are by no means unique) and returns a unique set of approximate quantum circuits that can be executed on hardware. Our multi-step optimization framework has several tuning parameters that allow users to fine-tune, e.g., tolerances for the expectation-value landscape, circuits for a specific set of input states, maximum circuit depths and gate counts.

Overall, the goal of this project can be formulated as an innovative yet challenging research objective and it is described as follows:

The main objective of this thesis is to generate hardware-executable approximate quantum circuits that perform the same functionality as their corresponding reference (textbook) circuits with the least number of possible resources (quantum gates). With this objective, we also expect to improve the reliability and performance of these executable quantum circuits in the challenging NISQ era.

A number of research questions have been formulated that can simplify the process of achieving the above-mentioned research objective as follows:

1. How the concept of approximate computing is incorporated in the field of quantum circuit generation?
   (a) How different is the proposed approximate executable circuit generation technique compared to the existing state of the art executable circuit generation techniques?
   (b) What are the advantages and disadvantages associated with the proposed circuit generation technique?
2. How is the proposed circuit generation technique made computationally more efficient using a genetic algorithm? How the flexibility of the genetic algorithm is used to generate a unique set of executable quantum circuits?
3. Do the circuits generated from the proposed approach have better reliability compared to the circuits generated from the state of the art quantum compilers? What are the reasons behind these improvements?

1.4 Thesis layout

The rest of this thesis is organized as follows. Chapter 2 gives an overview of quantum computing (Section 2.1), existing (approximate) circuit generation techniques in the literature (Section 2.6) and background information (Section 2.2 – 2.6).

Chapter 3 presents our multi-level optimization algorithm. The different algorithmic components of our multi-level optimization approach, i.e., the *rule-based approximate gate substitution* (Section 3.1), *continuous parameter optimization* (Section 3.2) and the *compiler-based*

optimization (Section 3.3) techniques are briefly discussed using suitable examples. In addition, considering certain benchmarks as example circuits, approximate circuits for both `Rigetti` (Figure 3.24) and `IBM` (Figure 3.25) platforms have been generated using the proposed approach.

Chapter 4 describes the role of the genetic algorithm in the proposed approximate circuit generation technique followed by the experimental evaluation with respect to the resources (quantum gates), noise, and execution time for a collection of quantum circuits in comparison with quantum compiler toolchains provided by `IBM` and `Rigetti` compilers in Chapter 5.

In Chapter 6, several limitations and challenges of the proposed algorithm are discussed. Conclusions of the thesis project are drawn in Chapter 7. This chapter will also include the answers to the research question. Finally, some potential ideas for improving the proposed algorithm and future recommendations towards approximate circuit generation have been presented in Chapter 8. Also, a case study with respect to the physical implementation of $CNOT$ gates is presented in Appendices (Chapter A).

2 Literature survey

2.1 Quantum computing

In the domain of quantum computing, qubits are the single most important entity that is used to store and process information in the quantum world. Similar to its classical counterpart (bits), qubits are physically viewed as a 2-state quantum mechanical system [31]. In practice, 2-state quantum mechanical systems can be easily realized through the spins of the electrons where the 2 states can be visualized as spin-up and spin-down according to the chosen convention [32]. Also, the polarization of photons, i.e., vertical polarization and horizontal polarization of photons can be considered to represent a 2-state quantum mechanical system according to the chosen convention [33]. Although there are a lot of similarities with respect to their representation, qubits and bits are significantly different from each other in the following aspects:

- Quantum superposition:

  Superposition is one of the two fundamental phenomena in the field of quantum mechanics [34] that has been incorporated in the quantum computing world. In classical physics, the concept of superposition was first realized as the sum (superposed) of 2 or more individual waves that would result in a new wave entity which is either due to the constructive or destructive interference of the parent waves [35]. When visualized in the computational world, superposition gives us the power to hold multiple (more) information in a single entity similar to how the resultant wave holds some sort of information of the parent waves. From the computation point of view, while classical bits (classical bit 0 and classical bit 1) can only exist in either one or the other state (0 or 1), qubits can be used to represent information as a simultaneous superposition of both the state (information can be 0 and 1 at the same time), e.g., the following equation represents a quantum state with equal superposition of qubit 0 and qubit 1:

$$|\psi\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \tag{2.1}$$

- Quantum entanglement:

  The concept of entanglement [36], is one of the most important quantum phenomena based on which quantum computing is highly dependent. As the name suggests, 2 entities are considered to be entangled if and only if any one entity cannot be fully described without the help of the other, i.e., the 2 entities are inseparable. One of the most well-known examples is the decaying of the sub-atomic particle like Quarks, Leptons, Bosons, etc. Due to some environmental factors, these sub-atomic particles may decay into pair of entangled offsprings [37]. If the state of one of the offsprings is known then we also roughly have the information of the other offspring since the offsprings are highly correlated to each other. Similar to the previous example, we can

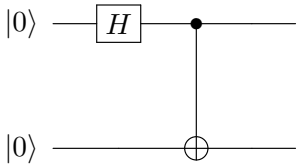


Figure 2.1: Preparation of the first bell state.

also consider the splitting up of a spin-0 electron due to some environmental factors [38]. If the state of one of the decayed electron is known to be spin-up then as a fact we also know the state of the other decayed electron to be a spin-down since both the decayed spins must add up to 0. The high correlation between the decayed particles of both, the electrons and the sub-atomic particles arises due to the necessary constraints of the conservation laws of physics [39].

In the quantum computing world, the concept of entanglement is used to communicate between the qubits without much interaction between them, e.g., if a quantum operation is applied to one of the entangled qubits, then it is highly possible that this would have affected the other entangled qubit due to its high correlation. Similar to the previous example, let us also consider the first quantum bell state that is generated using the circuit as shown in Figure 2.1. The final quantum state ($\frac{|00\rangle+|11\rangle}{\sqrt{2}}$) that is generated is a 2-qubit entangled state since both the qubits are inseparable, i.e., the qubits cannot be rewritten into individual product form. Also, if one of the entangled qubits are measured on the computational basis, the state of the other qubit is already known without even measuring it. Concepts of quantum entanglement and superposition cannot be realized using classical bits due to the restriction posed by the classical physics world. Therefore, quantum superposition and entanglement play a vital role in making the qubits very unique and powerful compared to their counterparts.

Apart from enhancing the properties of the qubits, the concept of quantum entanglement and superposition can be used to perform unique and interesting operations that are not possible to be realized in the classical world.

- Quantum teleportation:

  Is it possible to exchange information between 2 entities even without any direct interaction between them? In the quantum world, it is possible and it is termed quantum teleportation [40]. Consider the quantum circuit as shown in Figure 2.2. There are 3 qubits that are labelled using different colour strands indicating that they are different from one another. Initially, the qubits coloured green and red are entangled with each other. Later the blue and the green qubits are together measured on the first bell basis and the measurement outcomes are reflected on the qubits by the application of the *Pauli-Z* and *Pauli-X* quantum gates. Therefore, after the executing of the circuit, the red qubit holds the information which was earlier held by the blue qubit without having any direct interaction between them. In the classical world, it is not possible to exchange information (state of the bits) without having any direct interaction with each other.

- Quantum superdense coding:

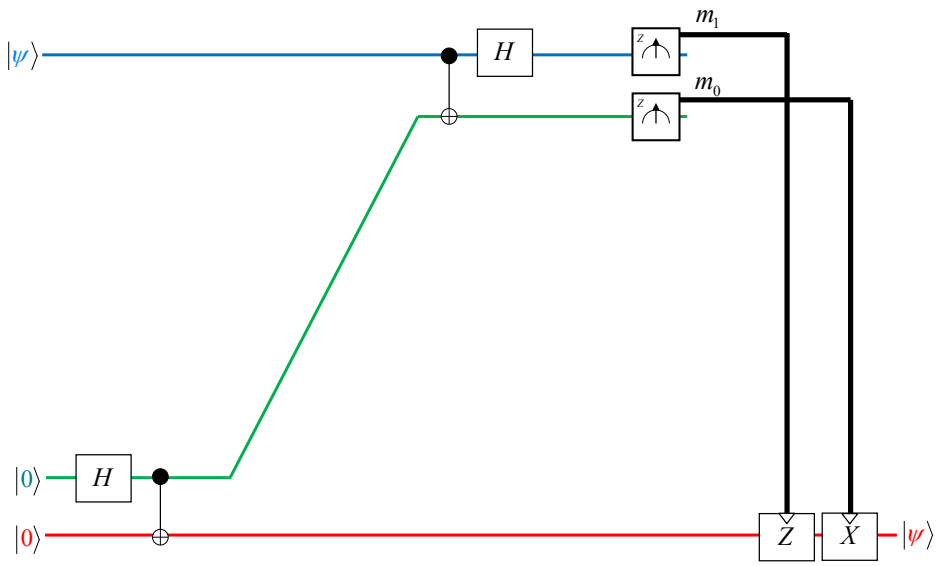
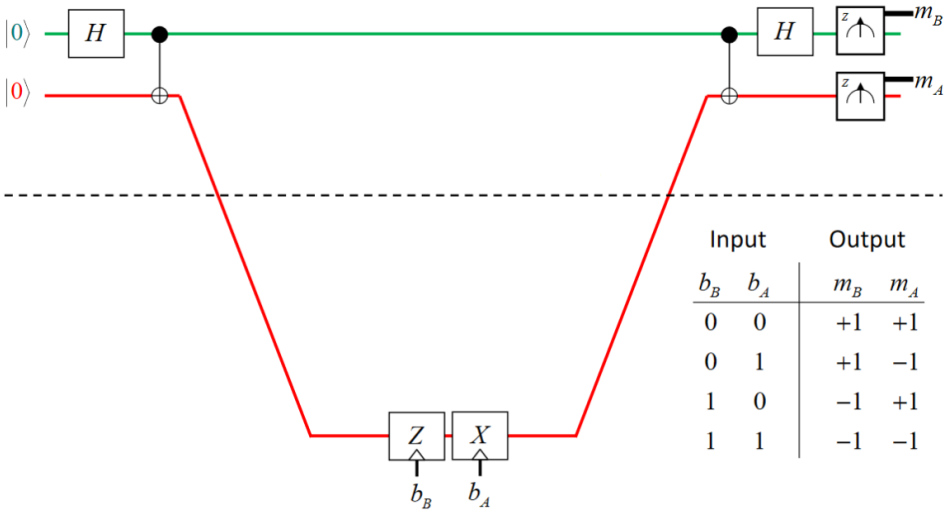


Figure 2.2: Quantum teleportation circuit [2].



Input		Output	
b_B	b_A	m_B	m_A
0	0	+1	+1
0	1	+1	−1
1	0	−1	+1
1	1	−1	−1

Figure 2.3: Quantum superdense coding circuit [2].

In the classical world, it is only possible to fully encode a single bit to a single qubit. However, it is not the same the other way around. As previously explained, qubits can hold more information. Therefore, a single qubit can be used to encode 2 classical bits using the quantum superdense coding technique [41]. Consider the quantum circuit as shown in Figure 2.3. There are 2 qubits that are indicated using different strands of colour. Initially, both green and red qubits are entangled with each other. Later, the 2 classical bits of information (b_A and b_B) are encoded using the *Pauli-Z* and *Pauli-X* quantum gates on the red qubit. Finally, both the qubits are measured on the first bell basis. Therefore, using the measurement outcomes (m_A and m_B), we can predict the 2

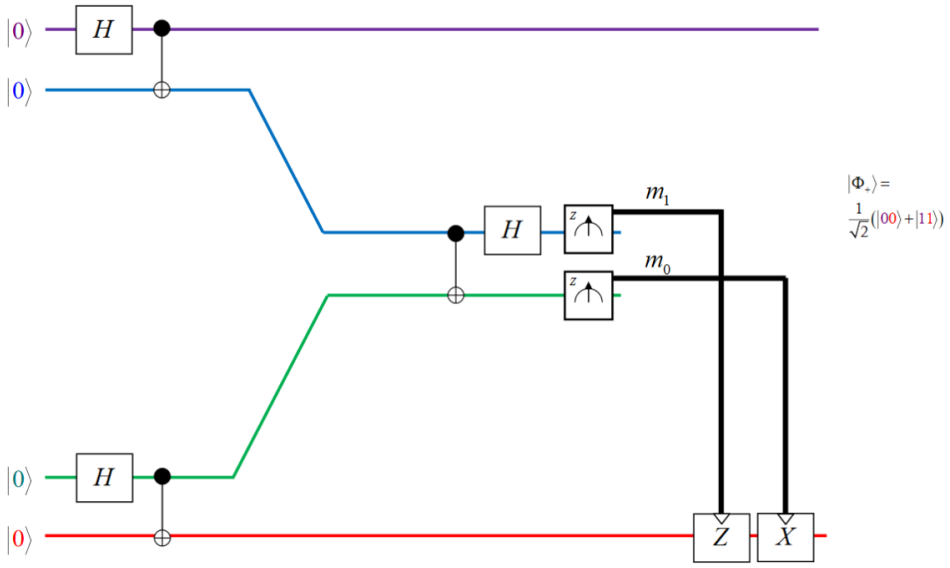


Figure 2.4: Quantum entanglement swap circuit [2].

classical bits of information that was encoded using the red qubit.

- Quantum entanglement swapping:

  Similar to the concept of quantum teleportation, here the qubits can be entangled to each other without having any direct interaction between them and this is possible using the quantum entanglement swapping technique [42]. Consider the 4-qubit quantum circuit as shown in Figure 2.4. All the qubits are colour coded to indicate that they are different from each other. Initially, both the purple and the blue qubits are entangled with each other. The same goes for the green and the red qubits. Later, the blue and green qubits are measured in the bell basis and their effect is reflected on the red qubit using the *Pauli-Z* and *Pauli-X* quantum gates. At the end of the execution, it is observed that the purple and the red qubits are entangled with one another without having any direct interaction between them. This is called entanglement swapping since the entanglement between the purple, blue and green, red qubits are swapped to the purple and red qubits.

As elaborated earlier, qubits are very unique and powerful entities that can be used to enhance the performance of computing like no other existing classical supercomputer. However, there are a lot of disadvantages associated with respect to the qubits that still hasn't been discussed yet.

Unlike their classical counterpart, qubits are very fragile, unstable and sensitive entities. They have a very short coherence time, i.e., qubits cannot hold or maintain the information for a very long time [22]. Due to several factors such as noise, temperature and other environmental factors and since they are so fragile, they tend to decohere to a mixed state [43]. Whereas the classical bits are very stable and unless voluntarily disturbed or defected during the manufacturing process, the state of the bit (classical information) remains unaltered. Therefore, in the quantum world, it is necessary to perform computation using less number

of operation so that the overall computation is not prolonged and performed as the qubits are still stable.

2.1.1 Qubit representation

In the classical world since bits are scalars they are just represented as 0 and 1. The general representation of a qubit is given by

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\ [9] \tag{2.2}$$

which is nothing but the linear combination of the basis vectors $|0\rangle$ and $|1\rangle$ which are orthogonal to each other. The complex numbers α and β are the probability amplitudes of $|0\rangle$ and $|1\rangle$, respectively. In addition, there is another condition that has to be satisfied and it is as follows:

$$|\alpha|^2 + |\beta|^2 = 1 \tag{2.3}$$

where $|\alpha|^2$ and $|\beta|^2$ are the probabilities of measuring state $|0\rangle$ and $|1\rangle$, respectively. But, unlike its classical counter part, qubits are vectors having a particular phase (+ or -) and magnitude associated to any given quantum state. Therefore, any quantum state can be represented as a vector. $|0\rangle$ state can be represented as $[1,0]^T$ and $|1\rangle$ can be represented as $[0,1]^T$. The $|0\rangle$ state is nothing but the *ket* notation of the 0 quantum state and is represented as column vector ($[1,0]^T$). Whereas, $\langle 0|$ state is the *bra* notation of the 0 state and is represented as a row vector ($[1,0]$ (conjugate transpose of $|0\rangle$)). In general the quantum state are represented using the conventional *bra-ket* notation. Since in the general representation, the states $\{|0\rangle\ ,\ |1\rangle\}$ are orthogonal to each other, they form an orthonormal basis state and are often referred to as the computational basis that span over the 2-dimensional hilbert space of the qubit [31]. Therefore, for a multi-qubit system, their orthonormal basis states are formed by the product of the individual basis states. As a result, the 2^n basis states of the a multi-qubit system spans over the 2^n dimensional hilbert space of the qubits, e.g., the orthonormal basis state for a 3 qubit system is given as follows:

$$\{|000\rangle\,, |001\rangle\,, |010\rangle\,, |011\rangle\,, |100\rangle\,, |101\rangle\,, |110\rangle\,, |111\rangle\} \tag{2.4}$$

where $|000\rangle = [1,0,0,0,0,0,0,0]^T$, $|001\rangle = [0,1,0,0,0,0,0,0]^T$ ..... $|111\rangle = [0,0,0,0,0,0,0,1]^T$

The Figure 2.5 depicts the Bloch sphere representation of a qubit [44]. In general, as we know that the qubit state is represented as $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$, in term of the Bloch sphere, its is represented as the following:

$$|\psi\rangle = e^{i\delta}(\cos(\frac{\theta}{2})\,|0\rangle + e^{i\varphi}\sin(\frac{\theta}{2}))\,|1\rangle \tag{2.5}$$

where θ is the polar angle $\in \{0,\pi\}$, φ is the azimuthal angle $\in \{0,2\pi\}$ and δ is the global phase. Therefore, by comparing the Equations 2.5 and Equantion 2.3, it i clear that $\alpha = e^{i\delta}(\cos(\frac{\theta}{2}))$ and $\beta = e^{i\delta}(e^{i\varphi}\sin(\frac{\theta}{2}))$. In addition, $e^{i\varphi}$ is the the relative phases of the qubit. It can also be observed that for single qubit system, the global phase upon measurement as no physical effect on the qubit. Therefore, the Bloch sphere representation a single qubit system can be further reduced to as follows:

$$|\psi\rangle = \cos(\frac{\theta}{2})\,|0\rangle + e^{i\varphi}\sin(\frac{\theta}{2})\,|1\rangle \tag{2.6}$$

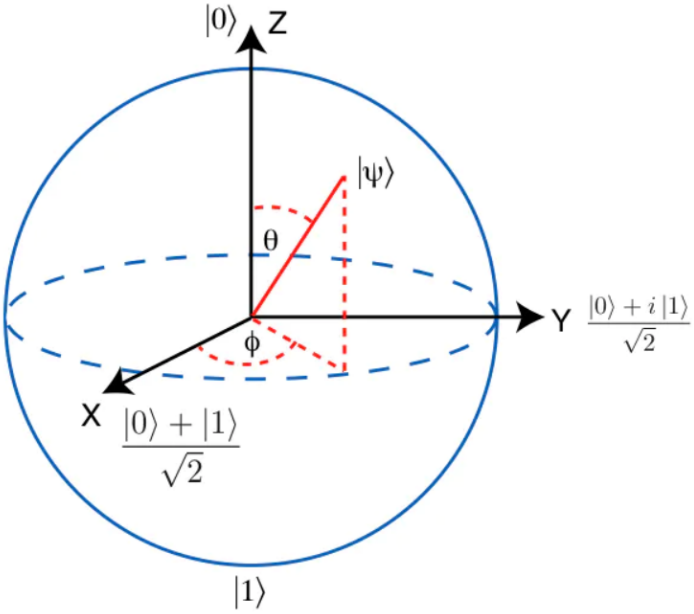


Figure 2.5: Single qubit Bloch sphere representation [2].

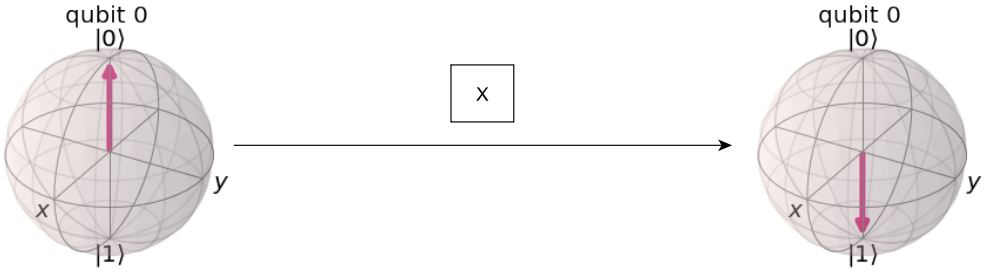


Figure 2.6: The action of the *Pauli-X* gate on the $|0\rangle$ state.

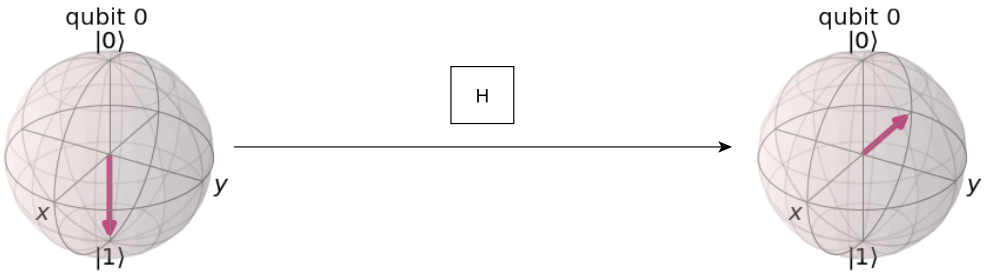


Figure 2.7: The action of the *hadamard* gate on the $|1\rangle$ state.

where α and β are now equal to $\cos(\frac{\theta}{2})$ and $e^{i\varphi}\sin(\frac{\theta}{2})$, respectively.

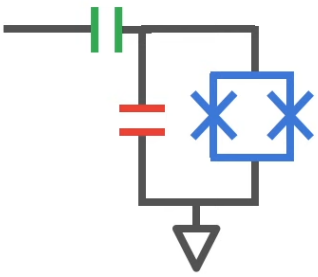


Figure 2.8: Electoral oscillators [3].

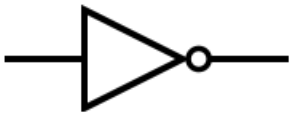
As discussed earlier, the { $|0\rangle$, $|1\rangle$ } form the computational basis (Z-basis) for a single qubit system. Apart from this, there are some other commonly used basis in a single qubit system such as X-basis which consists of { $|+\rangle$, $|-\rangle$ } orthonormal basis and Y-basis which consists of { $|+i\rangle$, $|-i\rangle$ } orthonormal basis. More importantly, in a single qubit system, the Bloch sphere also helps us to comfortably visualize the action of the quantum gates on the state of the qubit. Figure 2.6 shows the action of the *Pauli-X* gate on the $|0\rangle$ state. Since *Pauli-X* is nothing but a π rotation of the qubit around the X-axis, the state of the qubit is changed from $|0\rangle$ to $|1\rangle$. Figure 2.7 shows the action of the *hadamard* gate on the $|0\rangle$ quantum state. Since the *hadamard* gate is nothing but a $R_y(90)$ ($\frac{\pi}{2}$ rotation around the Y-axis) followed by $R_x(180)$ (π rotation around the X-axis),the state of the qubit is initially changed form $|0\rangle$ to $|+\rangle$ due to the R_y gate. Since the qubit is now in $|+\rangle$ state (positive end of the X-axis), the $R_x(180)$ (π rotation around the X-axis) does not affect the state of the qubit hence, remains in the $|+\rangle$ state.

2.1.2 Physical implementation

Now let us investigate the difference in the physical implementation between the classical bits and qubits. In the classical world, the classical bits are stored in the presence of absences of charge on a capacitor in a dynamic ram (DRAM) circuit [45]. So basically, if there is a charge stored in the capacitor then the bit is in logical 1 state else it is logical 0 state. Our logical bits 0 and 1 are made up of a lot of electrons. Hence, ensuring the stability state of the logical bits. But, a DRAM cannot be used to realized an existing quantum superposition since DRAM periodically checks the state of the capacitor to replenish the missing electrons (to maintain stability) if required. This checking can be performed using a photon which in case acts as nature. So, if there is some charge, the photon bounces off some unstable leaked electrons indicating that the bit is in logic 1 state else if the photon passes through without any interference, it indicates that the bit is in logic 0 state. As a result, this checking destroys the superposition as the photon can "observe" the state of the information and hence, collapses to either one of the logic states. Therefore, this and including the erratic nature of the electrons results in a high number of errors in the quantum world.

To be less susceptible to these quantum errors, we can use the concept of superconductivity to realize qubits [3]. In regular metals, like with a conventional DRAM circuit, every individual electron does its own thing. As electrons move around, they bounce off from the positively charged ions of the metal, radiating (leaking) vibrational waves that carry out the quantum information of the electron. Therefore, this generates a lot of quantum errors and

the information gets lost before we use it. However, when certain metals are cooled down, the electrons do not tend to scatter. As a result, we have control over the electrons. The quantum bits are just electoral oscillator [46] built from aluminium which becomes superconducting when cooled down to below 1° kelvin. When the oscillator is in the 0 state it has zero energy. When it is in the 1 state, it has a single quantum of energy. The 2 states of the oscillator with 0 and 1 quantum of energy are the logical states of the qubit. The electoral oscillators are shown in Figure 2.8. In this thesis, we will be considering the superconductive quantum systems of `IBM` and `Rigetti`. In particular, we will not concentrate on the physical implementation of the qubits rather, focus on the native gates that are supported by the corresponding systems. Apart from the superconductivity, the spins of the electron (spin-up and spin-down, irrespective of the convention), the polarization of a photon (vertical and horizontal polarization, irrespective of the convention), etc. can be used to physically realize a qubit [47].



Input	Output
0	1
1	0

Figure 2.9: *Not* gate and the corresponding truth table [4].



A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

Figure 2.10: *AND* gate and the corresponding truth table [4].

2.1.3 Quantum gates

In a classical computer, certain operations are applied on classical bits that change their state from one state to another. e.g., the *Not* gate as shown in the Figure 2.9 takes the classical bit 0 to classical bit 1 and classical bit 1 to classical bit 0. These operators are called logic gates. Similar to their classical counterpart, quantum computers are built using quantum logic gates. The quantum logic gates unlike the classical logic gates, map one unique state to another unique state, i.e., a quantum gate is a bijective function [48] in which every input state is uniquely mapped to an output state. Also, for any given quantum logic gate, the size (number of qubits) of any given input state is equal to the size (number of qubits) of any given output state. Whereas, for a classical logic gate like *AND* as shown in the Figure 2.10, the size (number of bits) of any given input state is not equal to the size (number of bits) of any given output state. Quantum gates can also be used to realize classical gate functions. Most of them can be realize using *Toffoli* quantum gate which will be explained later in this section, but in general, the irreversible classical logic gates are realized by the processes of embedding [49]. Since the quantum gates are bijective, they are invertible as well, i.e., when the gates are applied in reverse, the corresponding input quantum state is reconstructed. As a result, the quantum logic gates are reversible. In physical terms, quantum gates are applied to qubits by generating microwave signals of the appropriate frequency to change the state of the qubit (superconducting qubits in particular). Whereas, in the classical world, the classical logic gates are built using combinations of transistors. *MOSFETs* [21] are the commonly used transistors that are used to construct classical logic gates like *AND, NAND*, etc.

As discussed earlier, qubits can be represented as vectors and the quantum gates are bijective functions that uniquely map all the possible quantum states. Therefore, a quantum gate can be represented as a matrix of dimension 2^nx2^n that is applied on a nx1 representing a n-qubit state. In the following, a brief description of important quantum gates has been discussed.

- Pauli X:

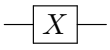


Figure 2.11: X gate.

Pauli X gate as shown in the Figure 2.11 is a single qubit operation that performs a π rotation around the X-axis. It changes the state of the qubit from:

$$\begin{aligned} |0\rangle &\mapsto |1\rangle \\ |1\rangle &\mapsto |0\rangle \end{aligned} \tag{2.7}$$

The matrix associated to this gate is given by:

$$PauliX = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = R_x(\pi)$$

- Pauli Y:

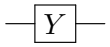


Figure 2.12: Y gate.

Pauli Y gate as shown in the Figure 2.12 is a single qubit operation that performs a π rotation around the Y-axis. It changes the state of the qubit from:

$$\begin{aligned} |0\rangle &\mapsto -i\,|1\rangle \\ |1\rangle &\mapsto i\,|0\rangle \end{aligned} \tag{2.8}$$

The matrix associated to this gate is given by:

$$PauliY = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} = R_y(\pi)$$

- Pauli Z:

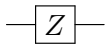


Figure 2.13: Z gate.

Pauli Z gate as shown in the Figure 2.13 is a single qubit operation that performs a π rotation around the Z-axis. It changes the state of the qubit from:

$$\begin{aligned} |0\rangle &\mapsto |0\rangle \\ |1\rangle &\mapsto -|1\rangle \end{aligned} \tag{2.9}$$

Therefore, it applies a negative phase if the state of the qubit is $|1\rangle$

The matrix associated to this gate is given by:

$$PauliZ = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = R_z(\pi)$$

- Hadamard:



Figure 2.14: *Hadamard* gate.

Hadamard gate as shown in the Figure 2.14 is a single qubit operation that is used to create equal super position to the staes in the computational basis. It changes the state of the qubit from:

$$\begin{aligned} |0\rangle &\mapsto \frac{|0\rangle + |1\rangle}{\sqrt{2}} \\ |1\rangle &\mapsto \frac{|0\rangle - |1\rangle}{\sqrt{2}} \end{aligned} \tag{2.10}$$

The matrix associated to this gate is given by:

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

The *Hadamard* gate can be further decomposed in the following ways:

$$\begin{aligned} H &\mapsto R_x(\pi)\, R_y\left(\tfrac{\pi}{2}\right), \\ H &\mapsto R_y\left(-\tfrac{\pi}{2}\right) R_x(\pi), \\ H &\mapsto R_z(\pi), R_y\left(-\tfrac{\pi}{2}\right), \\ H &\mapsto R_y\left(\tfrac{\pi}{2}\right) \mapsto R_z(\pi). \end{aligned}$$

Rotation operations are in general represented using the following equation:

$$\exp iA\theta = \cos(\theta)I + i\sin(\theta)A$$

where I is the 2 x 2 identity matrix and A is one of the Pauli gates, indicating the axis around which the rotation of angle θ should take place.

- $R_x(\theta)$:

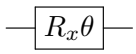


Figure 2.15: R_x gate.

$R_x(\theta)$ gate as shown in the Figure 2.15 is a single qubit rotation operation that performs a θ rotation around the X-axis.

The matrix associated to this gate is given by:

$$R_x(\theta) = \begin{bmatrix} \cos(\frac{\theta}{2}) & -i\sin(\frac{\theta}{2}) \\ -i\sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{bmatrix}$$

- $R_y(\theta)$:

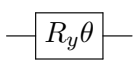


Figure 2.16: R_y gate.

$R_y(\theta)$ gate as shown in the Figure 2.16 is a single qubit rotation operation that performs a θ rotation around the Y-axis.

The matrix associated to this gate is given by:

$$R_y(\theta) = \begin{bmatrix} \cos(\frac{\theta}{2}) & -\sin(\frac{\theta}{2}) \\ \sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{bmatrix}$$

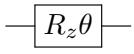


Figure 2.17: R_z gate.

- $R_z(\theta)$:

  $R_z(\theta)$ gate as shown in the Figure 2.17 is a single qubit rotation operation that performs a θ rotation around the Z-axis.

  The matrix associated to this gate is given by:

$$R_z(\theta) = \begin{bmatrix} e^{\frac{-i\theta}{2}} & 0 \\ 0 & e^{\frac{i\theta}{2}} \end{bmatrix} = e^{\frac{-i\theta}{2}} \begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix}$$

  since $e^{\frac{-i\theta}{2}}$ is the global phase, that can be ignored.

- SX($\sqrt{X}$):

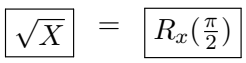


Figure 2.18: SX gate.

  SX($\sqrt{X}$) gate as shown in the Figure 2.18 is a single qubit operation that performs a $\frac{\pi}{2}$ rotation around the X-axis.

  The matrix associated to this gate is given by:

$$\sqrt{X} = \frac{1}{2} \begin{bmatrix} 1+i & 1-i \\ 1-i & 1+i \end{bmatrix} = e^{\frac{-i\pi}{4}} R_x(\frac{\pi}{2})$$

  where $e^{\frac{-i\pi}{4}}$ is the global phase difference between the SX ($\sqrt{X}$) and $R_x(\frac{\pi}{2})$

- S:

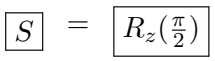


Figure 2.19: S gate.

  S gate as shown in the Figure 2.19 is a single qubit operation that performs a $\frac{\pi}{2}$ rotation around the Z-axis.

  The matrix associated to this gate is given by:

$$S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} = R_z(\frac{\pi}{2})$$

- T:

  T gate as shown in the Figure 2.20 is a single qubit operation that performs a $\frac{\pi}{4}$ rotation around the Z-axis.

$$\boxed{T} \;=\; \boxed{R_z(\tfrac{\pi}{4})}$$

Figure 2.20: T gate.

The matrix associated with this gate is given by

$$T = \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{4}} \end{bmatrix} = R_z(\frac{\pi}{4})$$

Multi-Qubit operations

- CNOT:

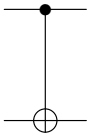


Figure 2.21: *CNOT* gate.

CNOT gate as shown in the Figure 2.21 is a 2-qubit operation that is used to perform a *X (Pauli X)* operation on the target qubit if and only if the control qubit is in the state $|1\rangle$.

It changes the state of the qubits from:

$$\begin{aligned} |00\rangle &\mapsto |00\rangle \\ |01\rangle &\mapsto |01\rangle \\ |10\rangle &\mapsto |11\rangle \\ |11\rangle &\mapsto |10\rangle \end{aligned} \tag{2.11}$$

where the most significant qubit is the control qubit and the least significant qubit is the target qubit.

the matrix associated to this gate is given by:

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

- CZ:

CZ gate as shown in the Figure 2.22 is a 2-qubit operation that is used to perform a *Z (Pauli Z)* operation on the target qubit if and only if the control qubit is in the state $|1\rangle$.

It changes the state of the qubits from:



Figure 2.22: CZ gate.

$$\begin{aligned} |00\rangle &\mapsto |00\rangle \\ |01\rangle &\mapsto |01\rangle \\ |10\rangle &\mapsto |10\rangle \\ |11\rangle &\mapsto -|11\rangle \end{aligned} \tag{2.12}$$

where the most significant qubit is the control qubit and the least significant qubit is the target qubit.

Therefore, it applies a negative phase on the target qubit only if the state of the control qubit is $|1\rangle$

the matrix associated to this gate is given by:

$$CZ = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

- Toffoli:

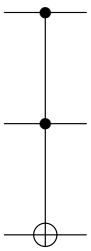


Figure 2.23: *Toffoli* gate.

Toffoli gate as shown in the Figure 2.23 is a 3-qubit operation that is used to perform a *X (Pauli X)* operation on the target qubit if and only if both the control qubits are in the state $|1\rangle$.

It changes the state of the qubits from:

$$
\begin{aligned}
|000\rangle &\mapsto |000\rangle \\
|001\rangle &\mapsto |001\rangle \\
|010\rangle &\mapsto |010\rangle \\
|011\rangle &\mapsto |011\rangle \\
|100\rangle &\mapsto |100\rangle \\
|101\rangle &\mapsto |101\rangle \\
|110\rangle &\mapsto |111\rangle \\
|111\rangle &\mapsto |110\rangle
\end{aligned}
\tag{2.13}
$$

where the most significant qubit and the middle qubit are the control qubits and the least significant qubit is the target qubit.

the matrix associated to this gate is given by:

$$
Toffoli = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{bmatrix}
$$

In addition to the reversible properties, the quantum logic gates have some additional properties.

- Unitary:

  A quantum gate U is said to be unitary if and only if $U^\dagger U = I$ where $U^\dagger$ is the complex conjugate transpose of the matrix U. All the aforementioned quantum logic gates are unitary matrices. Since quantum gates are invertible, the inverse of any quantum gate is given by $U^{-1} = U^\dagger$. Some of the quantum gate inverses are shown below:

  – $R_y(\theta)$ gate:

  $$
  R_y(\theta) = \begin{bmatrix} \cos(\frac{\theta}{2}) & -\sin(\frac{\theta}{2}) \\ \sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{bmatrix}
  $$

  $$
  R_y^{-1}(\theta) = R_y^\dagger(\theta) = \begin{bmatrix} \cos(\frac{\theta}{2}) & \sin(\frac{\theta}{2}) \\ -\sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{bmatrix}
  $$

  – S gate:

  $$
  S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} = R_z(\frac{\pi}{2})
  $$

  $$
  S^{-1} = S^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & -i \end{bmatrix} = R_z(\frac{-\pi}{2})
  $$

- Hermitian:

  A quantum gate H is said to be hermitian if and only if $H = H^{-1} = H^{\dagger}$, i.e. if a quantum gate is itself a self inverse gate. There are very few unitary quantum gates that are also hermitian and some of them are as follows:

  – Hermitian Unitary Quantum gate

$$\text{Pauli X} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = R_x(\pi)$$

$$\text{Pauli Y} = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} = R_y(\pi)$$

$$\text{Pauli Z} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = R_z(\pi)$$

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

  – Non-hermitian unitary quantum gates

$$S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} = R_z(\frac{\pi}{2})$$

$$T = \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{4}} \end{bmatrix} = R_z(\frac{\pi}{4})$$

  Therefore, all quantum hermitian gate are unitaries but not all quantum unitary gates are hermitian.

2.1.4 Quantum algorithms

Above we have briefly discussed about small quantum circuits (quantum gates). Large number of quantum gates can be used to construct complex quantum circuits and these large circuits are also sometimes called quantum algorithms. These quantum algorithms are inspired by real life applications and problems. The whole aim of implementing these algorithms on a quantum computer is to decrease the overall time complexity and resources. In thesis, we have considered the quantum algorithms :

- Deustch algorithm [50]:

  This is one of the first algorithms and in general, is used for benchmarking. In this quantum algorithm, we have been handed " quantum black box (also known as oracles) " [51] with multiple 1 bit to 1 bit Boolean function f_i encoded into unitary function U_{f_i} as shown in the Figure 2.24. The whole objective of this algorithm is to determine whether the function f_i is balanced or unbalanced. A function f_i is said to be balanced if there is an equal distribution between all the possible output states. The 1 bit *IDENTITY* function and 1 bit *NOT* function are typical examples of 1 bit balanced functions. The 1 bit *RESET* function and 1 bit *SET* function are typical examples of 1

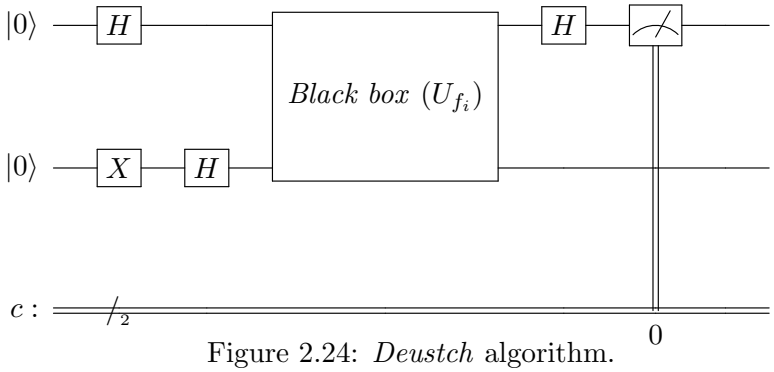


Figure 2.24: *Deustch* algorithm.

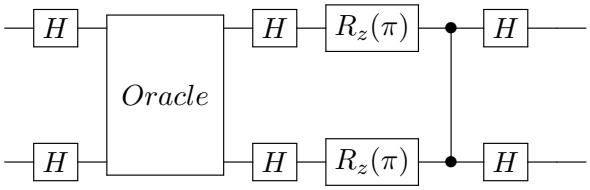


Figure 2.25: 2 qubit *Grover's* algorithm.

bit unbalanced functions. Classically, we would require to make 2 calls of the black box to determine whether the function is balanced or unbalanced. By using the quantum approach, we can figure out the function f_i is balanced or unbalanced by using only 1 call of the quantum black box.

- Bernstein-Vazirani algorithm [52]:

  Similar to the *Deustch* algorithm, we need to find out the value of an unknown secret string a of size n bits. This unknown secret string a is a part of the function $f(x) = a^T x \ mod \ 2$ which is encoded using unitary matrices to be implemented on a quantum computer. The encoded unitary matrix is nothing but the quantum black box (or oracle). Classically, we would require n calls of the classical black box to determine the secret string a. Turns out that by using the quantum approach, only 1 call of the quantum black box is required.

- Grover's algorithm [53]:

  Grover's algorithm is quite different from the *Deutsch* and the *Bernstein-Vazirani* algorithm. This algorithm has been inspired by a trivial search-based computational problem, e.g., if we were to search a particular entry in a huge directory, it would take us an enormous amount of time. Therefore, to tackle this issue, Grover proposed to encode this problem in the form of unitary matrices (quantum black boxes) so that it can be implemented on a quantum computer.

  Grover's algorithm is a search based quantum algorithm in which the whole objective is to find the one correct solution from a pool of 2^n possible states. This solution is again encoded using unitary matrices and is often referred to as the quantum oracles. Classically, it would require $O(2^n)$ calls of the classical oracle to search the right solution, whereas, by using the quantum approach we would just require $O(\sqrt{2^n})$ calls of the quantum oracle thus reducing the search space for, especially huge datasets. The

quantum circuit for a 2-bit *Grover's algorithm* is shown in Figure 2.25. Since $n = 2$, there are in total 4 oracles possible. We would require only 1 call of the quantum oracle to search for the right solution that is encoded using the quantum oracle whereas, we would require 3 calls of the classical oracle in the worst-case scenario to search for the right solution.

2.2 Classical optimization

Nowadays, the word optimization is commonly used in different fields of engineering, science and technology e.g., machine learning algorithms such as linear [54] and logistic regression [55], deep neural networks [56], program and compiler optimizations [57], circuit optimization in the field of electronics, structure optimization in the field of civil and mechanical engineering, etc. By definition, optimization involves minimizing or maximizing a particular cost function using tunable design parameters until an optimal or desired solution is obtained. In general, optimizers are often used to train mathematical models using some training data so that we can achieve a model that is nearly perfect in terms of the real world. From a computational point of view, there are plenty of optimizers that are in use namely:

- Gradient descent [58].
  - Stochastic gradient descent [59].
  - Mini-batch gradient descent [60].
- Momentum based gradient descent [61].
- Nesterov Accelerated Gradient [62].
- Adagrad [63].
- RMSProp [64].
- Adam [65].

Out of these classical optimizers, gradient descent (ascent) is most simple, efficient and the most commonly used optimizer in the world. The model parameters are updated using the following formula :

$$x^{(t+1)} = x^{(t)} - \eta \nabla_x f(x^{(t)}) \tag{2.14}$$

where $x^{(t)}$ is the current parameter that has to be updated to obtain $x^{(t+1)}$ and η is nothing but the step size. $\Delta f(x^{(t)})$ is the derivative or the gradient of the cost function $f(x^{(t)})$. These model parameters are optimized based on a given cost function. Throughout this thesis, we will be using the gradient descent optimizer. In addition, the derivative required to perform the gradient descent operations are calculated using automatic differentiation.

The concept of classical optimization is becoming very popular in the field of quantum computing and has also resulted in several new domains called quantum machine learning [66], hybrid quantum computation [67], quantum annealing [68], etc.

There are many ways by which the concepts of classical optimization and quantum computing can be mixed. Below is an illustration of a small example of how both concepts can be fused. Consider the circuit as shown in Figure 2.26. The whole objective is to

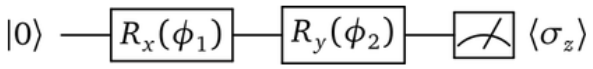


Figure 2.26: Optimization of rotation angles [5].

tune the rotation angles in such a way that the state of the qubit is changed from $|0\rangle$ to $|1\rangle$ ($|\psi\rangle = R_y(\phi_1)R_x(\phi_2)\,|0\rangle$). Using the expectation value of the qubit as a cost function ($\langle Z\rangle_{|\psi\rangle} = \langle\psi|Z|\psi\rangle$), we can optimize (maybe using gradient descent) the corresponding rotation angles to achieve the desired quantum state. Therefore by using the concepts of classical optimization we can train the parameterized quantum circuits to achieve a certain desired quantum state.

2.3 Expectation value

In the field of quantum computing, the expectation value of an observable A is the average value of all the corresponding measurement outcomes [69] and it is given as follows:

$$\langle A\rangle_{|\psi\rangle} = \langle\psi|A|\psi\rangle \tag{2.15}$$

where A is an observable using which the expectation value is calculated and $|\psi\rangle$ is an arbitrary quantum state. In a single qubit system, the expectation ranges form -1 to 1. Let us consider 2-dimensional computational or Z-basis states $|\psi_1\rangle = |0\rangle$ and $|\psi_2\rangle = |1\rangle$. The expectation value of the Z operation for the both, ψ_1 and ψ_2 is given by $\langle Z\rangle_{|\psi_1\rangle} = +1$ and $\langle Z\rangle_{|\psi_2\rangle}$ = -1. $\langle Z\rangle_{|\psi_1\rangle} = +1$ indicates that there is a 100% probability of measuring the state $|0\rangle$ whereas, $\langle Z\rangle_{|\psi_2\rangle}$ = -1 indicates that there is a 100% probability of measuring the state $|1\rangle$. In quantum computing, irrespective of the basis, the measurement outcomes can also be visualized in the form of the *eigen values* of the corresponding observable that is associated to a particular *eigen state* (basis states). In this case, for the state $|\psi_1\rangle = |0\rangle$, the eigen state obtained after the measurement in the computational or Z-basis is given by +1 eigen value whereas, for the quantum state $|\psi_2\rangle = |1\rangle$, the eigen state obtained after the measurement in the computational or Z-basis is given by -1 eigen value. Also, $\langle Z\rangle_{|\psi_1\rangle} = +1$ and $\langle Z\rangle_{|\psi_2\rangle}$ = -1 indicates that on average, the measurement (Z-basis) of the qubits would result in +1 and -1 eigen state of the Z operator, respectively. Let us consider $|\psi_3\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$. In this case, the expectation value of the Z operation for $|\psi_3\rangle$ is given by $\langle Z\rangle_{|\psi_3\rangle} = 0$. Let us understand why the expectation value $\langle Z\rangle_{|\psi_3\rangle}$ is 0. The state $|\psi_3\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$ is an equal superposition of 2 eigen states (in this case, states of the computational basis) $\{\ |0\rangle, |1\rangle\ \}$. If we made a very large number of measurements in the computational or Z-basis about half would result in +1 eigen state (i.e., $|0\rangle$) and other half of the measurement would result in -1 eigen state (i.e., $|1\rangle$). As a result, in this case, the average of the measurement outcomes in terms of the eigen value is 0. Since the expectation value helps us in distinguishing the quantum state one another in a scalar fashion, it is commonly used as a cost function to train parameterized quantum circuits to reach a desired quantum state.

2.4 Approximate computing

Approximate computing together with the help of classical optimization techniques has paved the ways for a lot of new optimizations. With the help of approximate computing, we can

- reduces the use of additional resources,
- reduce the energy dissipation (reduce energy costs),
- is as close as possible to the desired outcome, and
- improves the overall performance.

Approximate computing has already been in use for a very long time, e.g.,

- Approximate storage [70]: Truncating lower bits of floating-point data to accommodate more data.
- Approximate logic circuits [71]: Using these circuits helps us reduce overall hardware costs e.g., approximate adders, approximate multipliers, etc.
- Quantization [72]: Encode values to nearest acceptable numbers. e.g., the concept of quantization used in the working of analog to digital converter (ADC), etc.

The concept of approximate computing has made its way through the field of quantum computing as well. Especially, in the current NISQ era, approximation has gained a lot of recognition. One such example is the Quantum Approximate Optimization Algorithm (QAOA) [73]. The QAOA is a heuristic algorithm that has been used to solve NP-hard problems such as MAX CUT [74], MAX 3-SAT [75], QuASeR [76], QiBAM [77], etc. at least up to a certain extent. QAOA is inspired by classical approximation techniques and provides an optimal approximate solution [78]. The concept of approximation is introduced due to the probabilistic nature of the solution, i.e., the QAOA generates the expected solution with high probability. In addition, this algorithm is used to solve linear equations of systems as well.

The quantum HHL algorithm is also one of the prime examples that fall under the domain of approximation. The HHL algorithm is used to solve a system of linear equations ($Ax = B$) more efficiently on a quantum computer. It reduces time complexity from $O(Nsk\log(\frac{1}{\epsilon}))$ to $O(\frac{\log(N)s^2k^2}{\epsilon})$ where s,k,ϵ and N are the number of non-zero numbers, condition number of the system, accuracy of approximation and size of the matrix (A is NxN), respectively [79]. This significant reduction in the time complexity is due to the introduction of the approximation concept. While the classical algorithms always provide an exact solution (x), the quantum HHL algorithm in an approximate solution [80].

The concept of approximate computing is also used in many quantum circuit generation techniques. The parameterized quantum circuits form the basis for circuit recompilation or generation. The whole objective behind the concept of circuit generation is to construct a quantum circuit using available resources such that it is identical if not, close to the reference or target quantum circuit [81, 82, 83]. Initially, the parameterized circuits are designed in a way such that they are configured towards an identity matrix (therefore, a lot of structures are possible). Once the baseline circuit is set, the parameters of the various quantum gates are trained and updated using classical optimization techniques such as gradient descent. Classical optimization as the name suggests should be performed using classical computers,

whereas the quantum circuits (and the cost function) can be evaluated using simulation (ideal quantum computer) or by implementing them on a physical quantum device. These tuneable parameters are updated based on a cost function. There exists a number of cost functions but the most commonly used techniques are as follows:

- Unitary matrices: The distance/difference between the unitary matrix associated with the training quantum circuit and the unitary matrix associated with the target quantum circuit can be taken into consideration [81, 83]. It is also called the *trace* of the training and the target unitary matrix. Based on this cost function, the parameters of the quantum circuits are optimized to function exactly if not close to the target circuit. But, as the number of qubits increases, the time taken to compute the cost function increases exponentially.

- Expectation value: The difference between the expectation value or the average outcome of the training quantum circuit and the target quantum circuit can also be considered as a potential cost function [84]. The expectation value is more flexible compared to the unitary matrix since it can capture most of the circuit information using very little space. The parameterized circuit can be trained to be identical, if not close to the expectation value of the target circuit, therefore, resulting in an exact/approximate quantum circuit.

The above examples discuss how quantum computation together with the concept of approximation can be used to solve real-life applications more efficiently compared to the current classical algorithms. The approximation of the solution not only helps us reduce the time complexity, number of resources required, etc. but also has opened the doors for visualizing a new approach towards quantum computation.

Since the outcome of any quantum computation is highly probabilistic, the concept of approximate computing can be easily fused into it, i.e., instead of computing for an exact solution, we can perform the computation to find a probabilistic solution that is close to the exact solution provided that the usage of additional resources are reduced. In this thesis, the concept of approximate computing together with quantum computing has been visualized innovatively.

2.5 Genetic algorithm

Nature is involuntarily very intelligent. Through millions and billions of years, it has found ways to optimize its surroundings. Being inspired by this, computer scientists have developed optimization tools known as evolutionary algorithms. The most commonly used of the evolutionary algorithms is the genetic algorithm. Genetic algorithm is a meta-heuristic that is inspired by the natural selection concept of biological evolution [85]. Overall, the objective of the genetic algorithm is to find the most optimal solution within a search space that can be efficiently computed. The genetic algorithm comprises of different components which together provide a systematic procedure for its implementation and they are as follows:

- Initial population.

- Encoding.

- Selection.

- Crossover.
- Mutation.

A detailed explanation of the above components is given in Section 4.2. The genetic algorithm is widely used to solve combinatorial optimizations problems such as knapsack and the travelling salesman problem. These NP-complete problems are highly compute-intensive and require a huge search space exploration until we the most optimal solution. Therefore, the genetic algorithm can be used to reduce the overall search space and compute the most optimal solution for these combinatorial optimizations problems. Also, genetic algorithm is becoming very popular in the field of circuit design optimization which again, is highly compute-intensive.

To enhance the performance even further, the genetic algorithm can be integrated with the concepts of quantum computation. The inherent parallelism offered by quantum computation and the flexibility offered by the genetic algorithm can be combined to solve these combinatorial and other optimization problems more efficiently.

Quantum-inspired genetic algorithm [86, 87, 88] is an emerging concept that has gained a lot of interest in recent years. The basic idea behind the quantum-inspired genetic algorithm is to visualize the qubit encoding as chromosomes. Using the quantum superposition phenomenon, the problems which are encoded using these qubits can be evaluated simultaneously. The quantum logic operations can be used to facilitate the entire genetic algorithm e.g., the quantum gates can be used to change or update the state of the qubit.

Quantum-inspired genetic algorithm is essentially used to solve large combinatorial optimizations problems where exponential search space exploration is required e.g., travelling salesman and knapsack problem. In the case of the travelling salesman problem, most of the existing quantum-inspired genetic algorithms encode the distance between the cities in terms of the superposition of the qubits [87]. Similar to the travelling salesman problem, the knapsack problem can also be solved by visualizing all the items in the bag as qubits which are random superposition [88]. The quantum superposition of the qubits not only helps in reducing the search space but also helps us reduce the resources that are required to carry out the process.

In addition to the combinatorial problems, the genetic algorithm can also be used to generate and synthesize quantum circuits [89, 90, 91, 92]. Based on the unique entries of the target unitary matrix, the "fittest" quantum circuit can be selected from a pool of arbitrary quantum circuits (initial population). From an abstract point of view, the mutation and the crossover operation is performed by interchanging the necessary quantum operations either between the selected quantum circuits or within the selected quantum circuit. Overall, the whole objective is to efficiently find the most optimal quantum circuit that resembles the target unitary matrix from a pool of potential solutions.

Being inspired by the above-mentioned approaches, in this thesis, the concept of the genetic algorithm has been incorporated innovatively. Moreover, the genetic algorithm is used as an assisting tool in the process of approximate circuit generation.

2.6 Training parameterized quantum circuits

Although the fusion of quantum computing and classical optimization has led to the discovery of new algorithms and improved the performance of the current existing algorithms, it is still

in its infancy. Quantum computing together with classical optimization has the potential to make a lot of breakthroughs in the coming future. In particular, parameterizable quantum circuits is one of the best examples that comes under this domain. The parameterizable quantum circuits can be further classified into:

- Solving mathematical/application based problems:

  In this sub-domain, the parameterized quantum circuits are used to solve mathematical equations. The Variational Quantum Linear Solver (VQLS) is one of the prime examples that fall under this domain [93]. Similar to the quantum HHL algorithm [94], VQLS uses the concept of Variational Quantum Eigensolver (VQE) to solve a system of linear equations ($A|x\rangle = |b\rangle$). Although it is not faster compared to the quantum HHL algorithm, the flexibility offered by the parameterized quantum gates enables us to implement these quantum circuits on NISQ, hence providing better reliability.

- Solving machine learning-based algorithms:

  In this sub-domain, the parameterized quantum circuits are visualized as machine learning models [95]. Supervised machine learning techniques such as regressions can be solved used parameterized quantum circuits. Quantum computation allows us to encode the corresponding problem and evaluate the compute-intensive cost function more efficiently. Since computations can be performed probabilistically using quantum circuits, parameterized quantum circuits can also be used to perform unsupervised learning techniques such as generative modelling [96] more efficiently. Nowadays, Hybrid quantum-classical neural networks which are built based on parameterized quantum circuits are becoming very popular [97].

- Generating substitute/surrogate quantum circuit:

  In this sub-domain, the parameterized quantum circuits are trained and optimized until the target circuit if not, approximate of the target circuit is achieved [81, 82]. Similar to the example that was discussed previously, the parameterized unitary quantum gates are trained and updated based on a given cost function to achieve the desired target circuit. The circuit design parameters are updated using classical optimization techniques.

All the above-mentioned concepts were initially expressed using quantum born machines. Quantum Born machines [6] are derivations of quantum Boltzmann machines [98] which are designed to deal with classical data on a quantum circuit. Born machines introduce a superposition of all the possible states of the system to describe the probability distribution. Similar to machine learning, Born machines are trained to achieve a desired target quantum state. Figure 2.27 represents quantum Born machines and it can be described using the following steps:

- Initialization: The optimal solution that is to be obtained should be encoded in the form of the parameterizable rotation angles of the quantum circuit.

- State measurement: The resulting state after applying the rotation angles are recorded and measurement is applied on the recorded state.

- Sampling: Repeat step 2 several times to record the probability distribution of the resulting state.

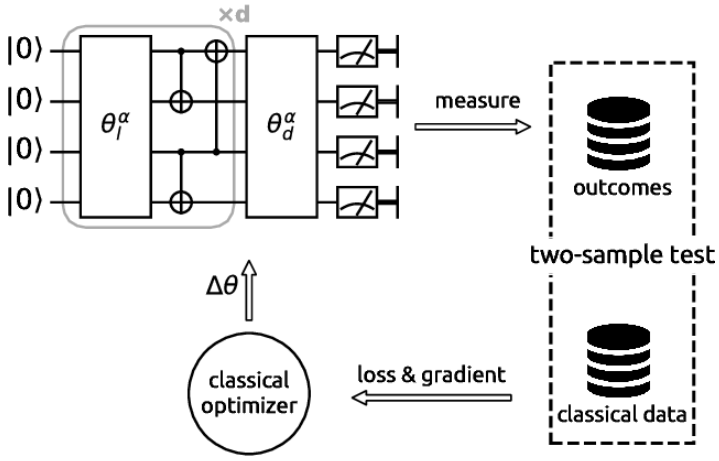


Figure 2.27: General representation of born machine [6].

- Parameter update: Use the mean square error (MSE) [99] as a cost function to calculate the difference between the measured and the target probability distribution (Sometimes expectation values can also be considered). The parameters are updated using classical optimizers such as gradient descent, etc.
- the above steps are repeated until we achieve the desired quantum state with the optimal parameters.

In this thesis, we have considered different kinds of literature which belong to the class of parameterized quantum circuits and in particular, we focus more on how these parameterized quantum circuits are trained to generate substitute or surrogate quantum circuits.

The proposed approach is more relevant to the work done in the paper [81]. This paper presents a variational hybrid quantum classical-quantum algorithm called quantum-assisted quantum computing (QAQC) which is used to train parameterised quantum unitary matrix V which is built using a combination of native gates of interest to be more close or similar towards the target quantum unitary matrix U. This paper uses the overlapping of the quantum unitary matrices ($\mathrm{Tr}(V^*U)$) as the cost function to train the unitary matrix V. In addition, the cost function considered is evaluated on real quantum hardware, therefore, taking the quantum errors into account as well. Although this paper takes into account the outcomes of a real quantum computer, it fails to elaborate on how the structure of the trainable unitary matrix V using the native gates of interest looks like.

This paper [82] represents a method for recompiling a quantum circuit of a unitary matrix A into the target unitary matrix B. Here the trainable unitary matrix, i.e., the parameterized unitaries are configured to the corresponding identity matrix. It trains the parametrised unitary matrix A into the target unitary matrix B. For optimization, this paper uses the adiabatic optimization technique, i.e., it uses quantum hardware to perform the energy minimization task. Although a general structure for training unitary matrix is given, it is still not elaborate as there is a high number of potential similar circuits possible.

This paper [95] provides an overview of hybrid quantum mechanical systems and discussed how parameterized quantum circuits can be visualized as machine learning models. In addition, a detailed review of applied quantum algorithm such as HHL, QAOA and VQE algorithms is given. In particular, this paper focuses on the hybrid approach of both classical

and quantum systems to speed up many data-driven machine learning applications. Since machine learning involves a lot of calculations which in general consumes a lot of time and resources, quantum computers can be used to speed up the process. Moreover, machine learning techniques such as supervised learning, generative models, etc. and their use in quantum computing has been described in detail in this paper.

This paper [83] proposes an algorithm for recompiling any given unitary matrix into a set of hardware supported native gates. In particular, this algorithm focuses on generating hardware supported surrogate circuits to reduce the overall count of $CNOT$ gates in the circuit. In addition, this paper also focuses on maintaining the qubit connectivity while synthesizing the $CNOT$gates. Similar to the previous state of the art approaches, this algorithm uses overlapping ($\mathrm{Tr}(V^*U)$) of the trainable unitary matrix V and the target unitary matrix (This also called the unitary distance between the U and V)

Both the papers [100] and [16] proposes a data-driven quantum circuit learning algorithm (DDQCLA) that can be used to benchmark NISQ devices and perform certain generative tasks. These generative tasks in particular are unsupervised learning tasks that produce the desired output just based on the regularities or pattern of the input data. Both the papers use a self-defined metric called qBAS, which is specifically used to evaluate the performance of the corresponding proposed algorithms on the canonical bars and strips dataset. The score was first proposed by [100]. The paper [16] uses Bayes and particle swarm optimization techniques in its algorithm and corresponding parametrize circuits are run on a trapped ion quantum computer. Whereas the algorithm performed in the paper [100] was performed on an IBM superconducting quantum processor.

Although all the approaches are unique in their own way, they have somewhat of the similar unclarities, inaccuracies and restrictions. Being inspired by these kinds of literature, this thesis:

- aims to provide a novel, clear, systematic and automated approach to create the baseline circuits that is trained to generate hardware-executable approximate quantum circuits that are identical to their corresponding reference circuits from a quantum hardware point of view.

- claims that the proposed approach would also work if implemented on a physical quantum system if not at the least on an ideal quantum computer.

- also aims to reduce the resources (in this case, the quantum operations/gates) that are required to generate the corresponding approximate circuits to reduce the susceptibility towards noise (environmental factors) and improving the performance while executing on a physical quantum system.

Generation of executable approximate circuits

3

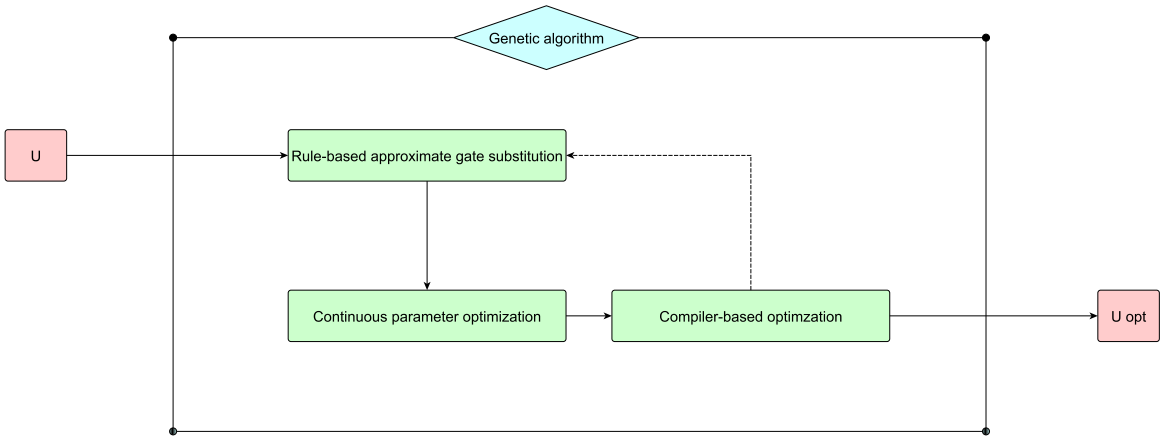


Figure 3.1: Flow chart of the multi-level optimization algorithm. The reiteration from circuit optimizer to *rule-based approximate gate substitution* (dashed arrow) is not yet included in our proof-of-concept implementation but supported conceptually.

In this chapter, we briefly describe the approximate circuit generation procedure. In particular, we will be focusing on generating executable circuits which will be able to run on physical/real quantum systems of `IBM` and `Rigetti` which are built based on superconducting electronics.

Let $\Psi = \{\alpha_0 |0\rangle^{\otimes n} + \cdots + \alpha_{2^n-1} |1\rangle^{\otimes n} \ : \ \alpha_i \in \mathbb{C}, \sum_{\ell=0}^{2^n-1} |\alpha_\ell|^2 = 1\}$ denote the infinite-dimensional set of all admissible quantum states for an n-qubit register $|\psi\rangle$ expressed in terms of the computational basis $\mathcal{B} = \{|0\rangle^{\otimes n}, ..., |1\rangle^{\otimes n}\}$. Furthermore, let the reference circuit be represented as unitary $2^n \times 2^n$ matrix U, which acts as the mapping between input and output states, i.e., $|\psi_{\text{out}}\rangle = U |\psi_{\text{in}}\rangle$.

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{3.1}$$

Equation 3.1 shows the unitary matrix for 2 qubit *controlled not gate* ($CNOT$). It can observed that there is an exact mapping between the basis states as shown below:

$$\begin{aligned} |00\rangle &\mapsto |00\rangle \\ |01\rangle &\mapsto |01\rangle \\ |10\rangle &\mapsto |11\rangle \\ |11\rangle &\mapsto |10\rangle \end{aligned} \tag{3.2}$$

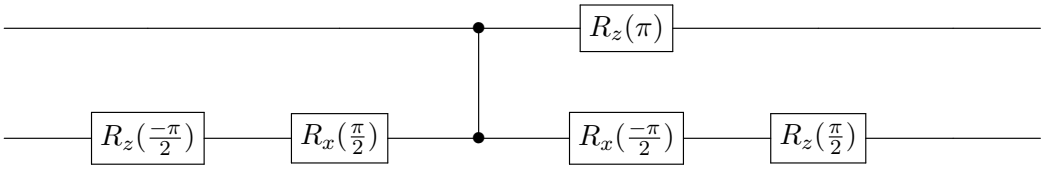


Figure 3.2: `Rigetti` compiler generated *CNOT* circuit.

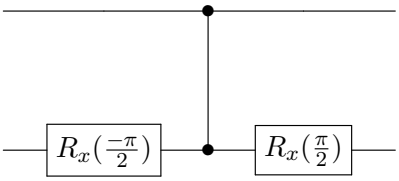


Figure 3.3: Approximate *CNOT* circuit generated using the proposed algorithm.

Figure 3.2, represents how a *CNOT* gate is implemented on a `Rigetti` quantum system thereby only using the native gates that are supported on a `Rigetti` device as shown in Table 1.1. Where as, on `IBM`, *CNOT* can be directly implemented without any further decomposition.

From the Figure 3.2, it can be observed that a total of 6 quantum gates is required to implement a *CNOT* gate which performs the same exact mapping (3.2) as the reference *CNOT* gate on a `Rigetti` quantum device. Whereas only one 2 qubit quantum gate is required to implement the *CNOT* operation on the `IBM` quantum system.

The overall objective of our proposed algorithm is to find a set of unique *hardware-executable circuits* that consists of native gates only, has minimal gate count and mimics the behaviour of the reference circuit for all basis states as good as possible. An example for the 2-qubit approximate *controlled-not (CNOT)* gate [101] is shown in Figure 3.3.

Figure 3.3, whose derivation will be detailed in Section 3.1 below. The approximate circuit in Figure 3.3, as expected, does not perform the exact mapping (3.2) as the reference *CNOT* gate.

The unitary matrix for the approximate circuit is given by the following:

$$Approx\ CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & i \\ 0 & 0 & -i & 0 \end{bmatrix} \tag{3.3}$$

This approximate circuit maps the basis state in the following way:

$$\begin{aligned} |00\rangle &\mapsto |00\rangle \\ |01\rangle &\mapsto |01\rangle \\ |10\rangle &\mapsto -i\,|11\rangle \\ |11\rangle &\mapsto i\,|10\rangle \end{aligned} \tag{3.4}$$

It is observed that the approximate circuit mimics the expectation value of the reference *CNOT* circuit. From the `Rigetti` hardware perspective, it is clear that approximate circuits

generated would readout or measure the same outcome for any given input state with the probability same as what we would expect from the reference $CNOT$ circuit. Therefore, on a physical device, we would only require 50% of the total number of `Rigetti` supported native quantum gates that are used to implement an exact $CNOT$ circuit.

In order to generalize the problem statement, let $\boldsymbol{\theta}_{\hat{U}} = (\theta_1, \ldots, \theta_N)^\top$ be the vector of parameters for the approximate circuit $\hat{U}$ and denote the set of all admissible unique approximate circuits with fixed size V by $\mathcal{U}_V$. The synthesis process can then be written as global optimization problem:

$$\hat{U}_{\text{opt}} = \underset{\hat{U} \in \mathcal{U}_V}{\operatorname{argmin}} \min_{\boldsymbol{\theta}_{\hat{U}}} \max_{|\psi\rangle \in \Psi} F(|\psi\rangle; U, \hat{U}, \boldsymbol{\theta}_{\hat{U}}), \quad V \to \min, \tag{3.5}$$

where the parameterized cost function is defined as follows

$$F(|\psi\rangle; U, \hat{U}, \boldsymbol{\theta}_{\hat{U}}) = \sum_{k=1}^{n} \left(\langle A^k \rangle_{U|\psi\rangle} - \langle A^k \rangle_{\hat{U}(\boldsymbol{\theta}_{\hat{U}})|\psi\rangle} \right)^2. \tag{3.6}$$

Here, A^k is an observable (can be $Pauli-Z, Pauli-X or Pauli-Y$) applied to the kth qubit and

$$\langle A^k \rangle_{P|\psi\rangle} = \langle (P\psi)^\dagger | A^k | P\psi \rangle \tag{3.7}$$

denotes its expectation value [69] of the state $|\psi\rangle$ upon application of the operator P. It is clear that the cost function equals zero when the approximate circuit mimics its reference counterpart *exactly* for all 2^n basis states.

Problem (3.5) is solved numerically using the multi-level circuit optimization algorithm which is sketched in Figure 3.1. It consists of different algorithm components which are chained together to generate a set of unique hardware-executable approximate quantum circuits to have the least possible circuit size. These algorithmic components will be briefly addressed in the forthcoming sections.

3.1 Rule-based approximate gate substitution

In this section, we will be elaborating on how the *rule-based approximate gate substitution* component of our multi-level circuit optimization algorithm will be implemented and also discuss the reasoning behind the approach used for implementing this type of algorithm, The limitations of this approach with a suitable solution will also be covered at the end of this section.

Let us consider the Bloch sphere for a single qubit system as shown in Figure 3.4 [7]. Let the qubit be initialized to the ground state $|0\rangle$ state. If a *hadamard* gate is applied to the qubit state $|0\rangle$, it rotates the qubit to the $|+\rangle$ state as seen in Figure 3.4. It is brought back to the original $|0\rangle$ state on the application of another *hadamard* gate as shown in Figure 3.5. From the above analysis, it is clear that a *hadamard* gates as 2 main functions namely:

- To change the original state (ground ($|0\rangle$) or excited state($|1\rangle$)) to an equal superposition state.
- To bring back the original state (ground $|0\rangle$ or exited state $|1\rangle$) from the equal superposition state.

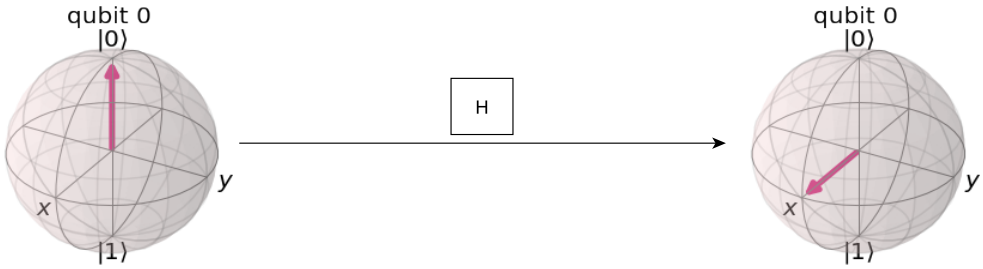


Figure 3.4: The action of the *hadamard* gate on the $|0\rangle$ state.

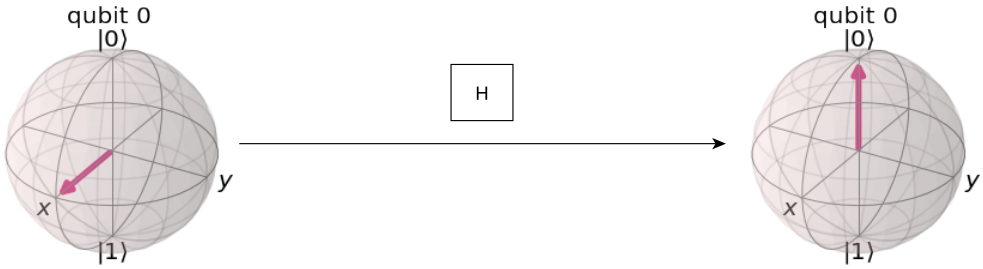


Figure 3.5: The action of the *hadamard* gate on the $|+\rangle$ state.

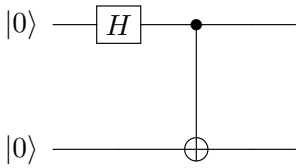


Figure 3.6: Bell state preparation circuit using *hadamard* gate.

But this is where 2 important observations can be made.

1. ***It is not necessary that both functions of the* hadamard *gates need to be satisfied at the same time for any given circuit.***

   It highly depends

   - on the functionality of the circuit for any given specific input state.
   - on the final state that is to be achieved after the execution of the circuit.

   Example: Consider the quantum circuit to prepare the first bell state as shown in Figure 3.6. For the given input state, the *hadamard* gate is only used to change the state of the first qubit $|0\rangle$ to $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$. This operation can also be performed by replacing the *hadamard* gate by $R_y(\frac{\pi}{2})$ [3] as seen in Figure 3.7 . Overall, both the circuits in the Figure 3.6 and Figure 3.7 achieve the same desired final quantum state.

[3] $R_y(\frac{\pi}{2})$ is also called a *Pseudo-hadamard* gate

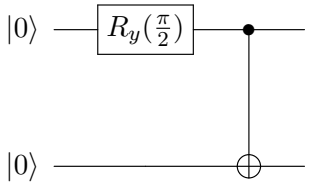


Figure 3.7: Bell state preparation circuit using R_y gate.

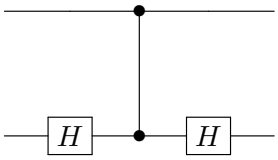


Figure 3.8: Exact $CNOT$ gate using *hadamards*.

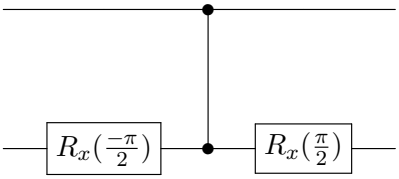


Figure 3.9: Approximate $CNOT$ gate.

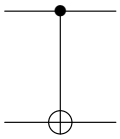
2. ***While implementing on a physical quantum device, in particular, the final state that is to be achieved does not matter as long as the measurement outcomes reflect what is to be achieved***.

   Example: Consider the quantum circuit as shown in Figure 3.8 that is used to perform a $CNOT$ operation. From the `Rigetti` hardware perspective, if we replace the *hadamard* gates by the corresponding rotation gates as shown in the Figure 3.9. It is observed that the measurement outcome for any given input state resembles the behaviour of the original $CNOT$ gates, i.e., both the circuits achieve the same desired state for any given input state.

Based on the above observations, we propose a *rule-based approximate gate substitution* algorithm as shown in Figure 3.1 of our multi-level circuit optimization algorithm. In the below description, we have considered the $CNOT$ gate being implemented on a `Rigetti` quantum system.

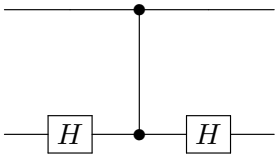
The first step of our algorithm consists of replacing all software-visible gates with approximate gate decompositions. Here and below we use the $CNOT$ gate depicted in Figure 3.10 as an example to illustrate the steps of this algorithm. Its decomposition into two *hadamard* (H) gates with an interjacent *controlled-Z* (CZ) gate is common practice [101] as is the substitution of H by one of the following rules [102]:

$$
\begin{aligned}
H &\mapsto R_x(\pi)\, R_y\left(\tfrac{\pi}{2}\right), \\
H &\mapsto R_y\left(-\tfrac{\pi}{2}\right) R_x(\pi), \\
H &\mapsto R_z(\pi)\, R_y\left(-\tfrac{\pi}{2}\right), \\
H &\mapsto R_y\left(\tfrac{\pi}{2}\right) R_z(\pi).
\end{aligned}
$$





Decomposed using *hadamard* gates





hadamard approximate gate substitution

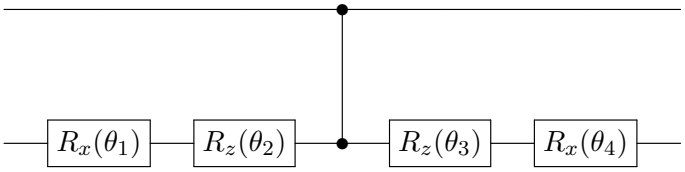


Figure 3.10: *Rule-based approximate gate substitution* for *CNOT* circuit.

Note that here and below the notation '$\mapsto$' represents the conventional mathematical representation of a combination of quantum gates being applied to a particular set of qubits, e.g, in our case, it should be understood as the literal substitution of the *hadamard* gate by a composition of two rotation gates, which is processed from right to left.

The problem of the above *exact* decompositions is that the rotation angles are fixed and leave no room for circuit optimization by adjusting the circuit's design parameters. A closer inspection of, say, the first substitution rule reveals that the application of $R_y\left(\frac{\pi}{2}\right)$ alone suffices to take $|0\rangle$ and $|1\rangle$ into the superposition states $|+\rangle$ and $|-\rangle$ [102], respectively. $R_y\left(\frac{\pi}{2}\right)$ and $R_x(\pi)$ together take care of the unitary and hermitian property of the *hadamard* gate.

It turns out that the following *approximate* decompositions

$$H \mapsto R_x(\theta) \quad \text{(Label 0)}, \tag{3.8}$$
$$H \mapsto R_y(\theta) \quad \text{(Label 1)}, \tag{3.9}$$
$$H \mapsto R_x(\theta_1)R_z(\theta_2) \quad \text{(Label 2)}, \tag{3.10}$$
$$H \mapsto R_z(\theta_1)R_x(\theta_2) \quad \text{(Label 3)} \tag{3.11}$$

often suffice to mimic the behaviour of the *hadamard* gate in practice and with the available native gate sets. For convenience, let us number the *hadamards* of the circuit column-wise from left to right. As will be demonstrated in the sequel, the choice $[3, 2]$ (first *hadamard* from the left is substituted by the approximate gate decomposition Label 3 (Equation 3.11) and the other *hadamard* is replaced by approximate gate decomposition Label 2 (Equation 3.10) depicted in Figure 3.10 leads to a valid surrogate circuit, whereas $[0, 1]$ and $[1, 0]$ does not. This is not a problem since inappropriate approximate circuits will be detected by the large values of the cost function (Equation (3.6)) and therefore be rejected by the topology optimization algorithm described in Section 4.2.

The substitution process moreover takes into account *generic* quantum gate identities [103] valid for arbitrary rotation angles, e.g.

$$HR_x(\theta)H = R_z(\theta), \quad 0 \leq \theta \leq 2\pi \tag{3.12}$$

to reduce the circuit's gate count. An exhaustive search over *all* possible substitutions yields 4^2 combinations for a simple $CNOT$ gate and becomes prohibitively expensive for realistic quantum circuits. We will present a genetic algorithm in Section 4.2 to replace exhaustive searches in practice. In this thesis, we will be focusing on replacing *hadamard gates* by the approximate decompositions as discussed earlier in the part of this section. In addition, we would expect that this procedure (proof of concept) would also work when employed for other software visible gates (high level specified). For the next subsections, we restrict ourselves to a single approximate circuit $\hat{U}$ with parameters $\boldsymbol{\theta}_{\hat{U}}$.

3.2 Continuous parameter optimization

In this section, we briefly describe the circuit optimization procedure. Also, we will be discussing the tweaks that have been incorporated in the commonly used optimizers to carry out the process more efficiently.

In general, the design parameters are optimized until the given cost function (Equation 3.6) is reduced to zero. The most common optimizer used in practice is the gradient descent optimizer [58] and in this thesis, we will be using the same optimizer to tune the circuit parameters which in our case is nothing but the corresponding rotation angles.

As mentioned earlier, in this thesis, we will be generating hardware-executable circuits in the paradigm of approximate computing. Approximate computing is also an emerging computing technology [30], but the concept is not new. Overall, the objective of approximate computing is to achieve a result as close as possible to what an exact computing technique would have resulted in with

- minimal use of resources (cost-effective).
- shorter physical execution times.

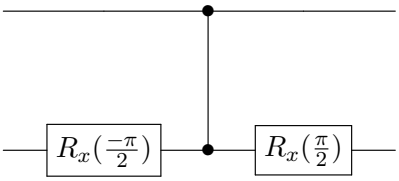


Figure 3.11: Approximate *CNOT* circuit with user-defined threshold = 0.

Inspired by this, in this thesis, with the help of classical optimization, we have visualized the concept of approximate computing in the field quantum computing using 3 different ways namely:

1. Generating hardware-executable quantum circuit whose *expectation value* (Equation 3.7) is identical to the corresponding reference circuit (Cost function = 0 (Equation 3.6)).

   Therefore, in this case, the circuit generated is an "approximate circuit" since the final state may or may not be equal to what the corresponding reference circuit would have resulted in. But the circuit generated would have the same measurement outcome as the corresponding reference circuit for any given input state.

   Example: Consider the hardware-executable approximate *CNOT* quantum circuit with Cost function = 0 as shown in the Figure 3.11. This circuit as zero tolerance (threshold = 0).

2. Generating hardware-executable circuit whose expectation value has some tolerable inaccuracies when compared to the expectation value of the corresponding reference circuit (Cost function > 0).

   In this case, as a result of these inaccuracies, fewer gates would be required to generate the hardware-executable circuit as compared to the circuit generated with an expectation value identical to the corresponding reference circuit.

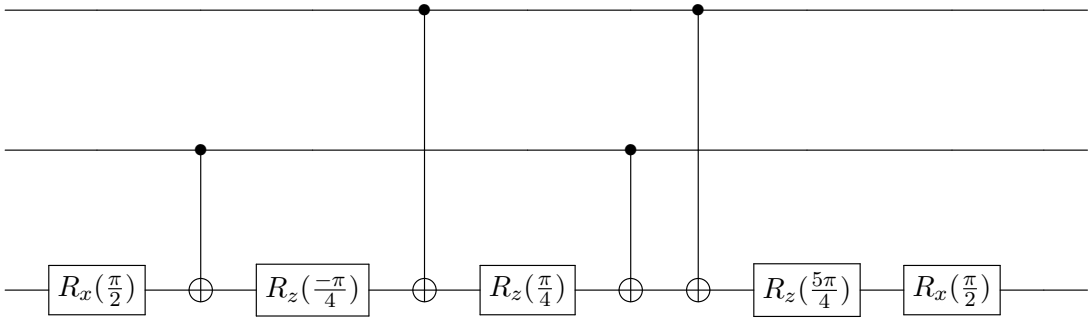


Figure 3.12: Approximate *Toffoli* circuit (for a `IBM` hardware) with user-defined threshold = 0.0857.

   Example: Consider the hardware-executable approximate quantum circuit with a user-defined threshold = 0.0857 (Cost function $\leq$ 0.0857) as shown in Figure 3.12. This circuit has a tolerance that is equal to the user-defined threshold.

3. Generating hardware-executable quantum circuits whose expectation value is identical

to the corresponding reference circuit for a specific set of input states, e.g., for the states of the computational basis.

Sometimes no given quantum circuit needs to work for all the inputs possible. It may be sufficient that it works for a set of specific input states (computational basis states). In this case, as a result, fewer gates would be required to generate the hardware-executable circuit as compared to the circuit generated that works for all the possible inputs.

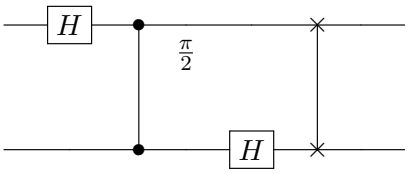


Figure 3.13: Reference *Quantum Fourier Transform* circuit.

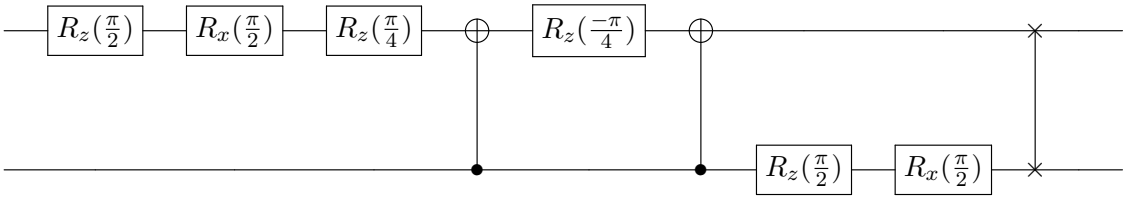


Figure 3.14: Approximate *Quantum Fourier Transform* circuit that works for any given input state.

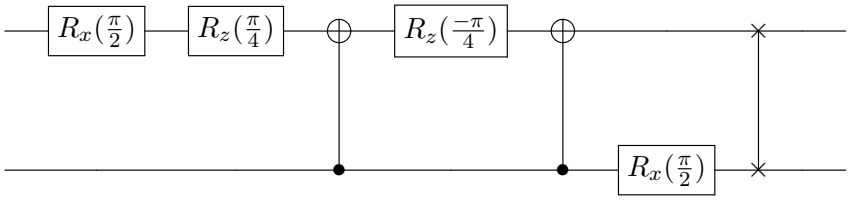


Figure 3.15: Approximate *Quantum Fourier Transform* circuit that works only for the computational basis states.

Example: Consider the 2-qubit *Quantum Fourier Transform* circuit, as shown in Figure 3.13. Figure 3.14 depicts the 2-qubit *Quantum Fourier Transform* circuit that works for all the possible input states whereas Figure 3.15 depicts the 2-qubit *Quantum Fourier Transform* that works only for the computational basis states ($|00\rangle, |01\rangle, |10\rangle, |11\rangle$). Both the circuits are generated using the proposed algorithm keeping in mind the `IBM` hardware supported native gates. It can be observed that fewer gates are required to implement *Quantum Fourier Transform* for a specific set of input states as compared to implementing *Quantum Fourier Transform* that works for all the possible input states.

Further in this section, we briefly explain how the continuous optimization procedure is carried out after the *rule-based approximate gate substitution* algorithm keeping the `Rigetti` hardware in mind.

The rotation angles θ_j of the approximate generated circuits resulting from the previous substitution procedure have not been specified yet and are considered the degrees of freedom

of a continuous design optimization problem to be solved in the second step for each individual circuit. Let $\hat{U}(\boldsymbol{\theta})$ denote the hardware-executable approximate circuit depicted in Figure 3.16 with the four design parameters $\boldsymbol{\theta} = (\theta_1, \theta_2, \theta_3, \theta_4)^\top$. Here and below, subscript $\hat{U}$ has been dropped for the sake of better readability.

Starting with either randomly initialized data or values close to the natively supported angles (e.g., $R_x\left(\pm k\frac{\pi}{2}\right)$ for $k \in \mathbb{N}$ for `Rigetti` quantum processors) the design parameter vector is updated successively using the gradient-descent optimizer [58], i.e.,

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \gamma \nabla_{\boldsymbol{\theta}} F(|\psi\rangle\,; U, \hat{U}, \boldsymbol{\theta}^t). \tag{3.13}$$

Here, γ is the user-defined step size and $\nabla_{\boldsymbol{\theta}}$ denotes the gradient of the cost function (3.6) with respect to the design parameters. The optimizer takes into account all $|\psi\rangle \in \Psi$ (typically all basis states or a user-defined subset thereof) simultaneously and returns a design vector that is optimal for all input states at the same time.

In addition to taking into account all the basis states, the optimizer should also be trained for a set of quantum states which are a superposition of all the basis states. This is required since benchmarks/circuits like *Quantum Fourier Transform* does not map an individual basis state to another individual basis state. Instead, it maps a single basis state to a linear superposition of all the basis states.

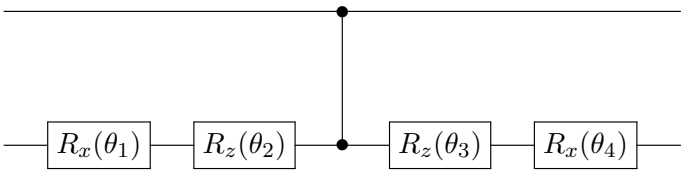


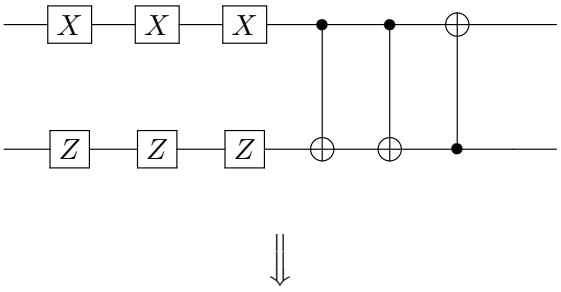
Figure 3.16: *CNOT* after *rule-based approximate substitution.*

In our implementation we use the PennyLane library [5] which allows to differentiate quantum circuits automatically and provides optimization and machine learning tools to determine the optimal design parameters for a finite set of input states $|\psi\rangle$ efficiently. The iteration procedure (Equation (3.13)) is stopped once the cost function value has dropped below a user-defined tolerance. For the example circuit depicted in Figure 3.16 indicates the approximate *CNOT* circuit with $\boldsymbol{\theta}_{\text{opt}} = \left(-\frac{\pi}{2}, -\frac{\pi}{2}, \frac{\pi}{2}, \frac{\pi}{2}\right)^\top$ and Cost function $= 0$, thereby turning it into an exact surrogate of the reference *CNOT* gate.

3.3 Compiler-based optimization

In this section, the *compiler-based circuit optimization* algorithm is going to be elaborated. In particular, the procedure is going to be described using various quantum circuits being implemented on a *Rigetti* quantum system.

By nature of continuous optimization, the design parameters are not given as symbolic values but as floating-point numbers. To carry out the compiler-based optimization steps described in the following, numerical data can either be converted to symbolic values or handled with a safety interval, e.g., all values x with $|x - \pi| < \text{tol}$ are identified with the symbol π similarly for $\frac{\pi}{2}$ and $\frac{-\pi}{2}$. With this convention in mind, our compiler-based circuit optimization performs the following steps:



Unitary gate cancellation

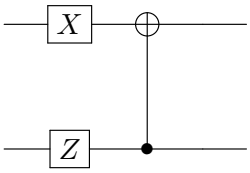


Figure 3.17: Unitary gate negation.

1. Collapsing of adjacent gates:

   - Cancellation of inverse gates, e.g.,

     Figure 3.17 shows how a unitary gate followed by its conjugate transpose can be negated.

     Figure 3.18 shows the cancellation of similar rotation gates with polar opposite angles.

   - Fusion of rotation gates, e.g.,

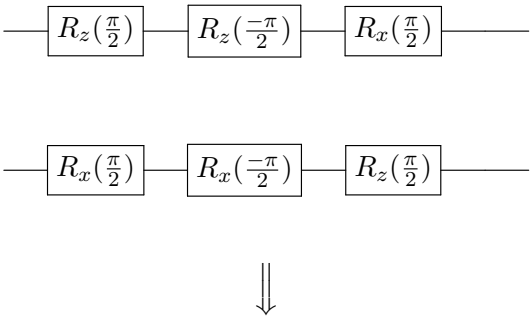
     Figure 3.19 shows the fusion of rotation gates of same kind to a single rotation gate. Hence, reducing the overall gate count.

   - Aggregation of gate groups using commutativity rules, e.g., [104].

     Figure 3.20 shows how the commute property of the quantum gates helps in further gate count reduction.

2. Substitution of *specific* gate identities [103], e.g.,

   - Figure 3.21 shows how the rotation gates sandwiched between the *hadamard* gates can be consolidated into single quantum operation with the help of commonly used gate identities such as:
     - $HR_x(\theta)H = R_z(\theta), \quad 0 \leq \theta \leq 2\pi$
     - $HR_z(\theta)H = R_x(\theta), \quad 0 \leq \theta \leq 2\pi$
   - In addition to the previous rule, in this algorithm, we have also considered substitution of generic identities, e.g., $R_y(\pi) = i\ R_z(\pi)\ R_x(\pi))$



Negation of polar opposite gates

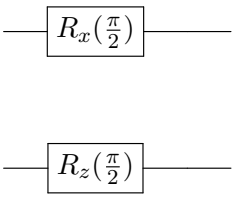
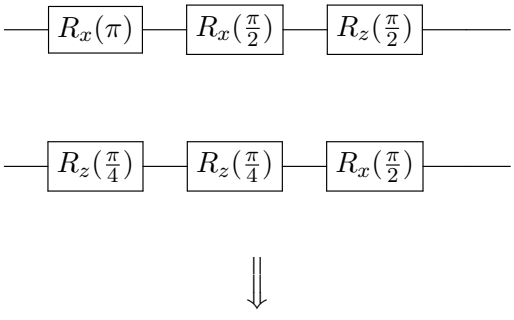


Figure 3.18: Negation of similar rotation gates.



Fusion of similar gates

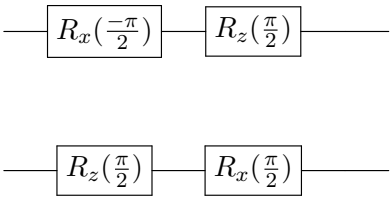
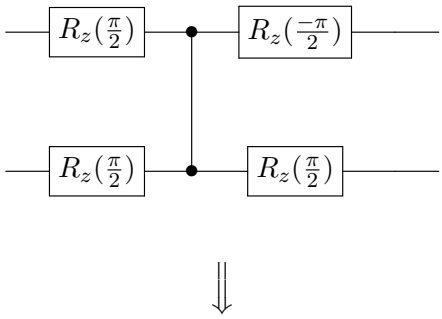


Figure 3.19: Aggregation of similar rotation gates.



Optimization using commutativity property

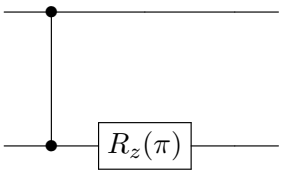
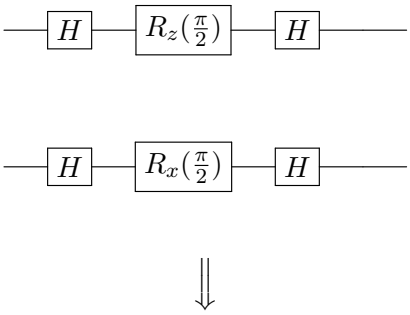


Figure 3.20: Gate count reduction using the commutativity property of the quantum gates.



Optimzation using *hadamard* gate identies

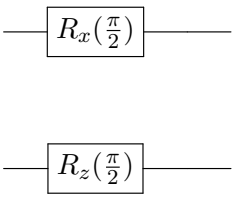


Figure 3.21: Specific quantum gate identities.

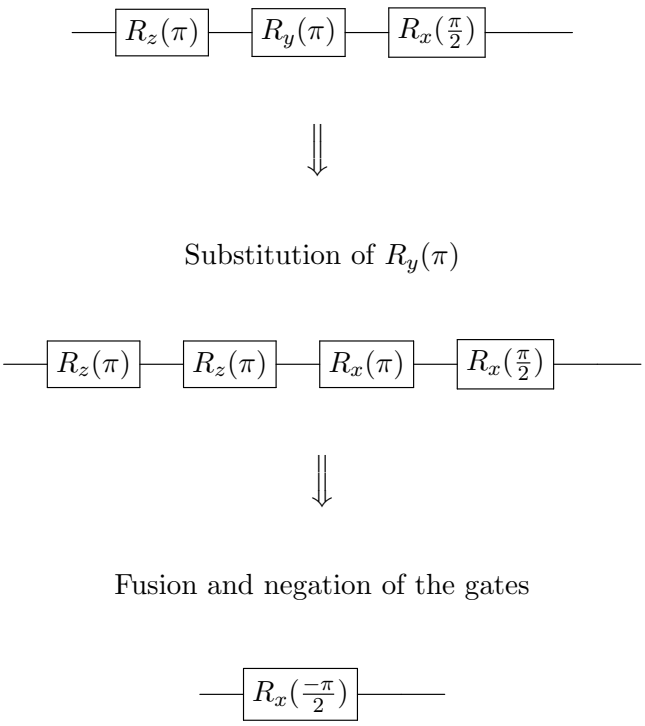


Figure 3.22: Generic quantum identities

With the help of these type of generic identities [101], the $R_y(\theta)$ gate substituted during the *rule-based approximate gate substitution* phase (Section 3.1) is "consumed" and therefore, helps in further reduction of the size of the quantum circuit. Figure 3.22 helps us describe this process more elaborately.

3. Optionally: Elimination of rotation gates whose design parameters have not changed compared to the initial values as their presence does not affect the circuit behaviour. This is further described using the Figure 3.23. Just before optimizing the circuit, it can be observed that those rotation angles are initialized to 1.5 radians. After several sweeps of running it through the optimizer, it is observed that some of the design parameters haven't been altered and as a result not affecting the circuit and are optimized to 0 radian and therefore can be eliminated.

The third rule requires some explanation and has its limitation as well. In our implementation we therefore initialize the rotation angles with values "close to" the expected values, e.g., $R_x(1.5)$. This, on the one hand, speeds up the convergence of the optimization process [61] and, on the other hand, ensures that some variation of the initial rotation angles occur unless the gate is not required and can be safely eliminated by the compiler-based optimization pass.

A gate, say, $R_x(1.5)$ or $R_x(-1.5)$ might be relevant for the correct functioning of the circuit while at the same time the compiler-based optimization would eliminate it as it is not supported on a `Rigetti` quantum system. Therefore, the remedy that we have incorporated is to run the circuit several times through the optimizer to ensure that the unaltered gates can be eliminated with confidence.

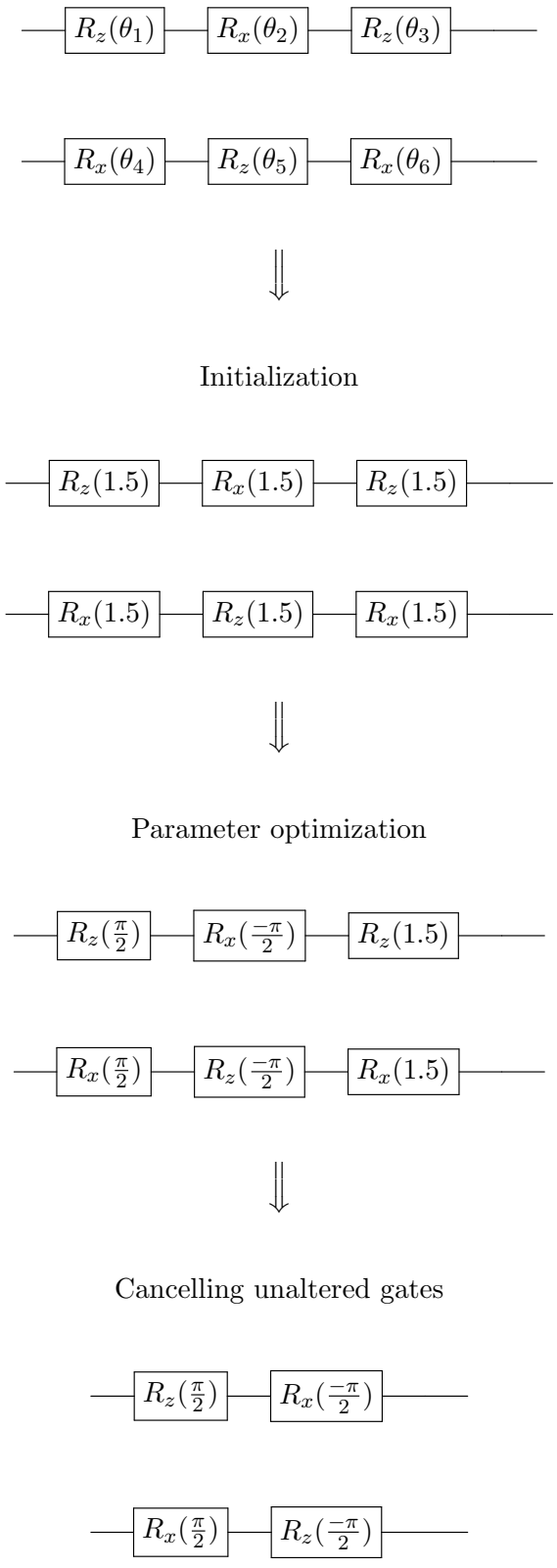


Figure 3.23: Negation of unaltered parameterized quantum gates.

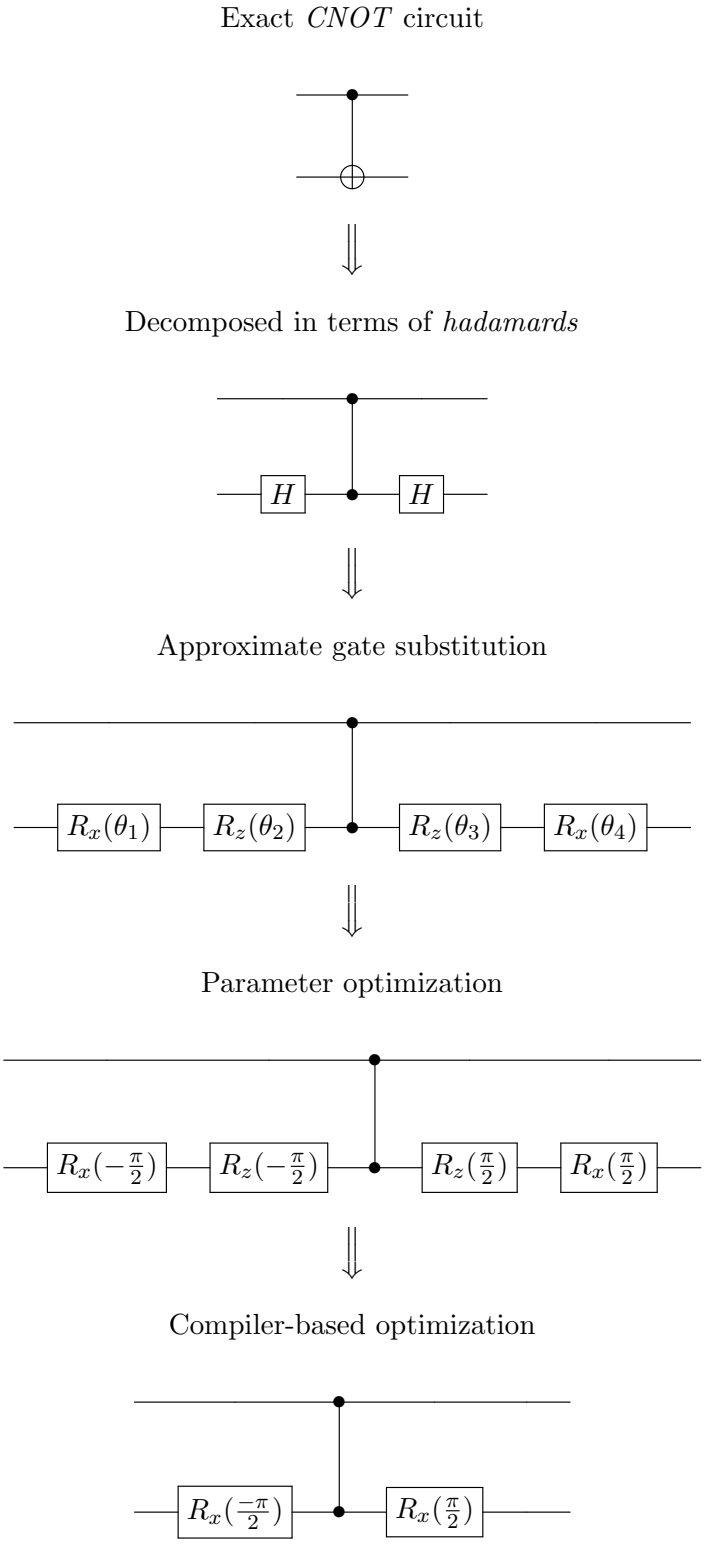


Figure 3.24: Approximate `Rigetti` executable $CNOT$ circuit generated from the proposed algorithm.

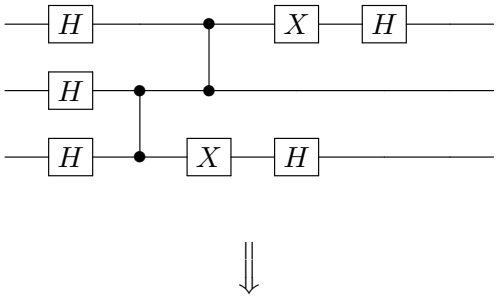
Any of the above modifications might enable further optimizations with adjacent gates not possible in previous rounds so that multiple sweeps are performed until no further changes are observed. In our experience, compiler-based optimization leads to a significant reduction of the circuit's depth and gate count.

Figure 3.24 illustrates the entire process (*rule-based approximate gate substitution, continuous parameter optimization and compiler-based optimization*) to generate an approximate *CNOT* circuit that can be implemented on a `Rigetti` platform..

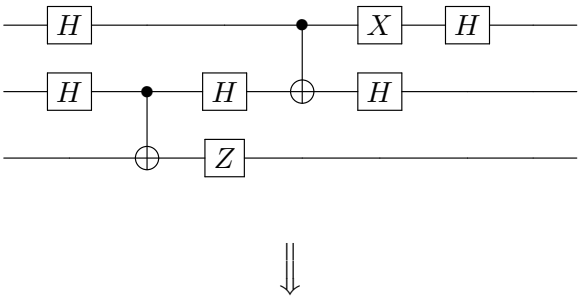
3.4 Approximate circuit generation for `IBM` quantum device

In this section, as an example, the process to generate an executable from the reference *BM1* quantum circuit [105] using the proposed algorithm is depicted. In this case, the proposed algorithm is used to generate an approximate quantum circuit that can be physically implemented on a `IBM` quantum device. Figure 3.25 illustrates this entire process.

Exact *BM1* [105] circuit.



Decomposed in terms of *hadamards* an *CNOT*



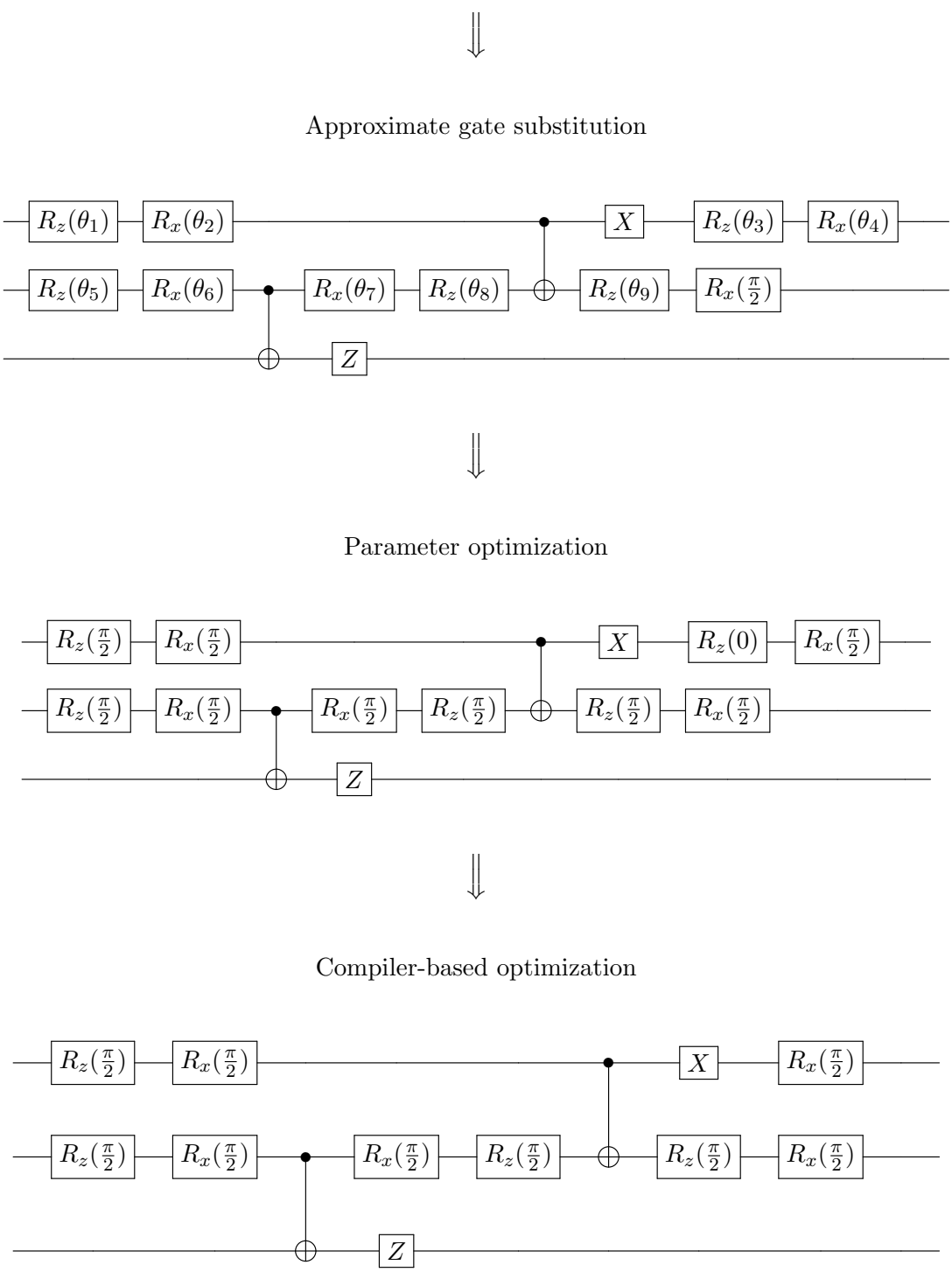


Figure 3.25: Approximate `IBM` executable *BM1* circuit generated from the proposed algorithm.

Genetic algorithm for circuit generation

4

4.1 Introduction to genetic algorithm

In this chapter, we will be elaborating on how the genetic algorithm [85] is incorporated in our approach and explain how this natural selection technique helps us speed up and efficiently carry out the whole process. In addition, we will be briefly discussing every component involved in the implementation of the customized genetic algorithm as shown in Figure 4.1. This notion of natural selection technique is inspired by Charles Darwin's theory of natural evolution [106]. Therefore, in general, a genetic algorithm selects the fittest, healthiest and optimal individual (solution) out of a huge pool of individuals (possible solutions).

As with every algorithm, the genetic algorithm is broken down using different steps and it is as follows:

1. Initial population:

   The initial population consists of all the individuals who have the potential to be the fittest amongst them all. From the programming perspective, every individual (possible solution) can be visualized as a string of binary bits (0's and 1's). From the biological point of view, the string can be viewed as a chromosome and the corresponding 0's and 1's in the string represent the genes of the chromosome which is nothing but the features associated with the corresponding individual. Further, in the brief explanation of this algorithm, we will be considering the strings of 0's and 1's as the possible solutions and the corresponding 0's and 1's labellings as the features of these solutions.

2. Encoding:

   Since the initial population is huge, some sort of compression is required to carry out future processes more efficiently. Therefore, the encoding phase is responsible for that. From the programming point of view, encoding can be implemented using different ways [107] and it is as follows:

   - Binary encoding.
   - Permutation encoding.
   - Real value encoding.

   A variant of both, the binary encoding and permutation encoding is implemented in our genetic algorithm.

3. Selection:

   The whole motive of the selection process is to select the fittest individuals (fit solutions) based on a specific fitness criterion. In programming, the fitness criteria are implemented as a fitness function based on which each possible solution will be selected. The fitness

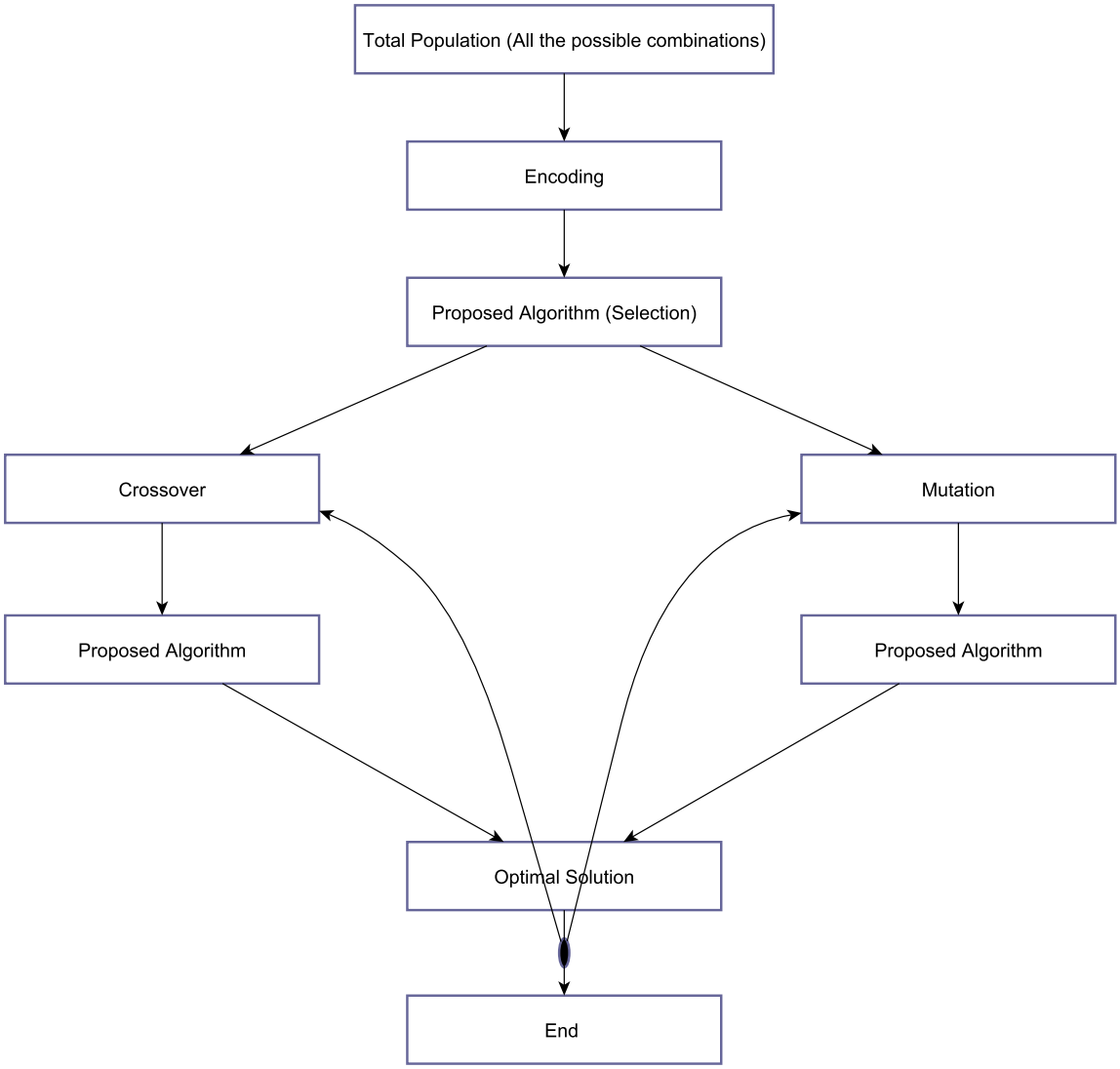


Figure 4.1: Flow chart of our genetic algorithm.

function varies based on the objective that is to be achieved at the end of the algorithm. This phase forms the basis for producing the future fitter individuals (solution).

4. Crossover [108]:

   Crossover is one of the most important components of a genetic algorithm. As the name suggests, crossover involves 2 fit individuals (parent chromosomes) interchanging their features (genes) to produce (mate) individuals (offsprings) who are possibly fitter than the 2 parent individuals. From the programming point of view, it is just interchanging the features (0's and 1's) of the parent solutions (a string of 0's and 1's) to produce a set of new solutions (a new string of 0's and 1's). Crossover can be implemented in multiple ways and it is as follows:

   – Multiple-point crossover techniques [109].

- Uniform crossover techniques [110].

We have implemented the uniform crossover technique in our customized genetic algorithm.

5. Mutation [111]:

   The mutation is also one of the most significant phases of the natural selection process. Unlike the crossover operator, the mutation is undergone within the individual itself. It is caused due to the interchanging of the genes within the chromosome (individual) and as a result, a new individual with different characteristics is formed. Although mutation is an intimidating phrase, in the computing world, we use the good side of mutation to overall increase the fitness of the individual (better solution). Again from the computing point of view, the mutation is performed by interchanging the features (0's and 1's) within the parent solution (strings of 0's and 1's) to produce a set of new solutions (new strings of 0's and 1's). Mutation operation can be performed using several ways and it is as follows:

   - Random shuffle operation [112].
   - Bit-flip operation (for binary encoded).
   - Reverse operation [113].

   We have implemented the mixture of both, the random shuffle and reverse operation to produce unique fit solutions in our genetic algorithm.

Both, the crossover and the mutation results in the next generation of individuals and are considered to be fitter and healthier compared to their previous generation. Both the processes are carried out until the most optimal individuals (solutions) are obtained. Genetic algorithm is flexible in many ways, i.e., it does not have any exact approach and can be implemented serially or parallelly based on the user specifications and configurations. Therefore, this enables us to customize the genetic algorithm to fulfil our needs and to achieve our ground objective.

4.2 Optimal circuit using the genetic algorithm

In this section, by exploiting the flexibility of the genetic algorithm, we are going to briefly explain how the algorithm can be visualized in terms of a circuit/gate-level design problem and discuss how the algorithm is incorporated in our multi-level framework. Moreover, the entire procedure is going to be further explained using suitable examples keeping both, the `IBM` and `Rigetti` hardware in mind.

From the *rule-based approximate gate substitution* point of view, obviously, an exhaustive search over all possible gate substitutions in the first step of our algorithm is only possible for very small circuits. Already a moderate-size circuit with 100 *hadamard* gates would require an exhaustive search over 4^{100} variations, which is more than 10^{36} times the estimated number of stars in our observable universe (300 sextillions) and thus impossible to solve by a brute-force approach. As a remedy, we resort to a genetic algorithm [85] approach as shown in Figure 4.1 which adopts a meta-heuristic strategy inspired by natural selection to efficiently explore the search space in rounds.

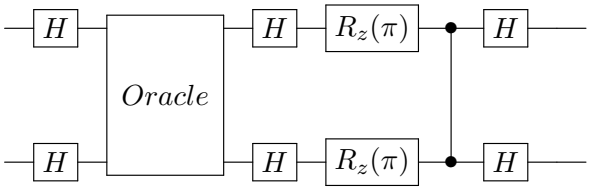


Figure 4.2: 2 qubit *Grover's* algorithm.

As an example throughout the rest of the section, Let us consider 2-qubit Grover's algorithm [53] depicted in Figure 4.2, whose 6 *hadamard* gates give rise to 4^6 possible topology configurations. The oracle can be interpreted as a user-definable callback function that will give rise to additional gates, which are, however, not *hadamard* gates and are tacitly ignored in what follows. For convenience, by numbering the 6 *hadamard* gates column-wise from top left to bottom right, this selection can be easily visualized as an integer vector, e.g., $[3, 1, 2, 3, 0, 1]$, which makes the implementation of *crossover* and *mutation* operations straightforward as shown in Algorithm 1 and Algorithm 2, respectively. In the case of our algorithm, we will visualize this integer vector of *rule-based approximate gate substitution* labels as a potential fit individual (chromosome) where each labelling (genes or features of the circuit) specifies the approximate decomposition by which the corresponding *hadamard* gate is being replaced with.

In the beginning, all possible gate substitutions are considered to be in the initial population and are declared generation zero. For the 2 qubits *Grover's* algorithm, our initial population consists of 4^6 potential approximate circuits that are created from the reference circuit by applying the four substitution rules (Equation 3.8 – 3.11) per *hadamard* gate. This phase indicates the initial population component of our genetic algorithm as shown in Figure 4.1.

Later, if required, a subset of all the potential approximate circuits (initial population) $\hat{U}$ is randomly selected. In our case, we select 1024 surrogate circuits that are created from the reference circuit by randomly applying one of the four substitution rules per *hadamard* gate. The threshold for selecting the subset is user-defined as it depends on the performance of the computing system of the user. In our case, we have considered a threshold of 5 *hadamard* gates, i.e., if n is the total number of *hadamards* gates, then the total number of potential approximate circuits selected is given by the following:

$$Subset = \begin{cases} 4^n & \text{if } n \leq 5 \\ 4^5 & \text{if } n > 5 \end{cases}$$

This indicates the encoding phase of our genetic algorithm as it is sort of a "compression" technique that is used to reduce the search space of our algorithm. Each of these selected circuits are optimized independently and at best in parallel using the multilevel framework work (*rule-based approximate gate substitution*, *continuous parameter optimization* and *compiler-based optimization*) as discussed throughout the Sections 3.1 – 3.3. This describes the selection phase of our genetic algorithm. The fitness function that we have considered to select the fittest solution (combinations) is nothing but the cost function as explained in Equation 3.6. The values of the cost function evaluated for the fully optimized circuits serve as a criterion to measure the circuits' fitness.

A few fittest circuits are carried over to the next generation without modification to secure

the reached fitness level for all times. The rest of the next generation is created by performing crossover [108] and mutation [111] operation on the fittest circuits. Both crossover and mutation operator generate new possible approximate circuits using the existing solutions. The newly generated which are potentially the fitter circuits are again passed through the multi-level framework (*rule-based approximate gate substitution,* continuous parameter optimization and *compiler-based optimization*) as shown in Figure 4.1.

The crossover operation operates on 2 parent circuits and creates two new offsprings by using the uniform crossover techniques with random masks [110]. This step reflects the crossover component of our genetic algorithm. The whole purpose of this operation is to recombine the good features of the parent individuals to produce a more healthy and fitter individual.

Algorithm 1: Crossover Operation

for *Every* (P_1, P_2) *of fit circuits from the parent generation* **do**
  Create random masks M_1, M_2 ; `// Bitstring of length 6.`
  for *Every bit in masks* **do**
    `/* First offspring */`
    O_1[i] = (M_1[i]==0 ? P_1[i] : P_2[i]);
    `/* Second offspring */`
    O_2[i] = (M_2[i]==0 ? P_2[i] : P_1[i]);
  end
end

Example: Let $P_1 = [3, 1, 2, 3, 0, 1]$ and $P_2 = [2, 3, 0, 1, 1, 0]$ be a pair of fit circuits from the parent generation and denote the two randomly generated masks by $M_1 = [0, 1, 1, 0, 0, 1]$ and $M_2 = [1, 0, 1, 0, 1, 0]$, respectively. The two offsprings generated by Algorithm 1 are then given by $O_1 = [3, 3, 0, 3, 0, 0]$ and $O_2 = [3, 3, 2, 1, 0, 0]$, whereby "? :" denotes the ternary if-then-else operator.

The mutation operation operates on a single parent circuit and creates two new offsprings by permuting the integer vector randomly and reversing the order of substitutions, respectively. This step represents the mutation operator of our genetic algorithm. These mutation operations are meant to keep the population more diverse and enhance the chances of better search space exploration.

Algorithm 2: Mutation Operation

for *Every fit circuit* P *from the parent generation* **do**
  `/* First offspring */`
  O_1 = RandPermute(P);
  `/* Second offspring */`
  O_2 = Reverse(P);
end

Example: Let $P = [3, 1, 2, 3, 0, 1]$ denote a fit circuit from the parent generation. A possible first offspring is given by $O_1 = [3, 3, 1, 1, 0, 2]$, where the second offspring equals $O_2 = [1, 0, 3, 2, 1, 3]$.

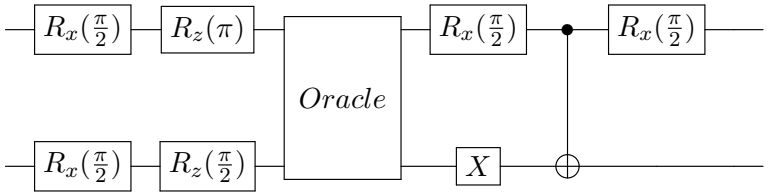


Figure 4.3: Approximate 2 qubit *Grover's* algorithm generated to be performed on a `IBM` quantum device.

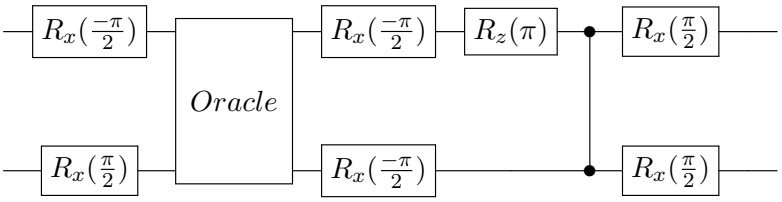


Figure 4.4: Approximate 2 qubit *Grover's* algorithm generated to be performed on a `Rigetti` quantum device.

Multiple rounds of this evolutionary algorithm are performed until no further improvements are obtained or the maximum number of iterations has been reached. From the final generation, the circuit with the lowest gate count that also has a cost function value below the user-specified threshold is selected as the global optimum to problem (3.5). Apart from exploring the search space efficiently, the flexibility of the described genetic algorithm allows the entire procedure to implemented in parallel since each potential approximate circuit that is selected from the initial pool are independent of each other, hence potentially increasing the efficiency of the computation.

Overall, with help the of the genetic algorithm embedded within our proposed framework, we can generate a set of optimal hardware-executable approximate quantum circuits for any given reference circuit using the least possible resources (least number of quantum gates).

Figure 4.3 and Figure 4.4 depicts the exact surrogate(approximate) circuits of the 2 qubit *Grover's algorithm* generated through the multi-level optimization approach for both, the `IBM` and `Rigetti` hardware, respectively.

Experimental evaluation and discussion

5

In this chapter, we apply our multi-level optimization approach to a collection of software-visible multi-qubit gates and quantum circuits to evaluate its performance in comparison with two state-of-the-art quantum compiler toolchains implemented in `IBM`'s qiskit framework [7] and `Rigetti`'s pyQuil library [18] that are used regularly for production runs. In particular, we will be focusing on the (a) the amount of resources (quantum gates), (b) reliability (noise susceptibility) and (c) execution time (duration of the physical implementation) related aspects of the quantum circuits. All tests were conducted on a consumer laptop with Intel Core i7-7700HQ Quad-Core Processor and 8GB GDDR5 RAM based on a prototype implementation of our approach in the PennyLane library [5]. The specific versions of the software libraries and toolchains used in our study are as follows:

- PennyLane version 0.14.0 [5],
- PyQuil / Forest SDK version 2.20.0 [18], and
- Qiskit version 0.23.2 [7].

`IBM`'s `qiskit.compiler.transpile` class provides three optimization levels controlled by the function `optimization_level`. Level-1 optimization collapses adjacent gates that negate each other. Level-2 optimization does inverse gate cancellation as the previous level and, in addition, uses the commutativity properties to enable further reduction in the gate count. Level-3 optimization includes all level-2 optimizations and in addition, performs re-compilation of sub-blocks of the quantum circuit to aggregate over sets of gate combinations. Since the level-3 optimizer is an aggressive optimization process (heavy and time-consuming), the circuits generated are not always deterministic. Therefore, 500 runs were performed for each benchmark from which the best results are reported in Table 5.1 and Table 5.3 and compared to the outcome of our circuit generation approach.

However, the `Rigetti`'s compiler, which is invoked by the function `qc.compiler.quil_to_native_quil`, offers a single optimization level which collectively performs all the optimizations considered by `IBM`'s compiler (transpiler [114]).

In order to generate approximate circuits for both, the `IBM` and the `Rigetti` hardware, certain adjustments to the proposed multi-level optimization algorithm is necessary, e.g., change the set of gate group aggregation rules, etc. Moreover, since our approach does not yet take the device-specific lattice [8] structures into account, we have considered the full qubit connectivity option provided by the `IBM` and `Rigetti` compiler toolchains.

5.1 Resource reduction

5.1.1 Quantum circuits without oracles

In our study, we have considered a collection of quantum circuits with and without oracle functions. Table 5.1 and Table 5.2 summarizes the oracle-free circuits that we have considered

Table 5.1: Resource evaluation of our approach for quantum benchmark circuits generated for the `IBM` platform.

Quantum circuit	No. of qubits	Optimization level	Size of the quantum circuit	
			IBM	Ours
BM1	3	1	25 (48.00%)	13
		2	24 (45.83%)	
		3	29 (55.17%)	
QFT	2	1	12 (8.33%)	11
		2	12 (8.33%)	
		3	42 (73.80%)	
QFT	3	1	20 (104.76%)	21
		2	20 (104.76%)	
		3	30 (30.00%)	
QFT	4	1	39 (2.56%)	38
		2	39 (2.56%)	
		3	55 (30.90%)	
QFT	5	1	57 (1.75%)	56
		2	57 (1.75%)	
		3	73 (23.28%)	
Toffoli	3	1	18 (44.44%)	10
		2	18 (44.44%)	
		3	18 (44.44%)	
2 bit Quantum Adder	4	1	40 (40.00%)	24
		2	38 (36.84%)	
		3	43 (44.18%)	
4 bit Quantum Adder	8	1	122 (39.34%)	74
		2	116 (36.20%)	
		3	131 (43.51%)	
8 bit Quantum Adder	16	1	286 (39.16%)	174
		2	272 (36.02%)	
		3	299 (41.80%)	
MC3X	4	1	46 (21.73%)	36
		2	46 (21.73%)	
		3	102 (64.70%)	
MC4X	5	1	94 (22.92%)	76
		2	94 (22.92%)	
		3	150 (49.33%)	

as benchmarks for the `IBM` and `Rigetti` platforms, respectively. *Controlled-NOT* (*CNOT*) [101], *swap* [115], *Toffoli* [116] and *multi-qubit controlled X (MC3X and MC4X)* [117] are basic quantum gates that are widely used in larger quantum circuits, whereas the *BM1* [105], *Quantum Fourier Transform (QFT)* [118] and the *quantum adders* [119] are examples of stand-alone quantum circuits.

In general, the quantum gate counts presented in Table 5.1 and Table 5.2 indicate that our multi-level optimization approach can reduce the number of native gates significantly and,

Table 5.2: Resources evaluation of our approach for quantum benchmark circuits generated for the `Rigetti` platform.

Quantum circuit	No. of qubits	Size of the quantum circuit	
		Rigetti	Ours
CNOT	2	6 (50%)	3
SWAP	2	14 (35.715%)	9
QFT	2	18 (11.12%)	16
QFT	3	45 (31.11%)	31
QFT	4	80 (23.75%)	61
QFT	5	118 (22.03%)	92
Toffoli	3	33 (54.54%)	15
2 bit Quantum Adder	4	64 (34.37%)	42
4 bit Quantum Adder	8	197 (33.00%)	132
8 bit Quantum Adder	16	474 (34.17%)	312
MC3X	4	90 (15.55%)	76
MC4X	5	195 (15.89%)	164

hence, lead to smaller executable circuits, in comparison with the optimized circuits produced by the vendors' toolchains. The specific gains, however, largely depend on the concrete circuit, its gates and the device-specific native gate set.

From the size of the quantum circuit presented in Table 5.1 and Table 5.2 it can be observed that our multi-level optimization approach leads to a significant reduction (up to 50% reduction) of the number of native gates that is required to physically implement the benchmarks compared to all the optimized circuits produced by the state of the art `IBM` and `Rigetti` compilers. These improvements, however, largely depends on the concrete circuit, its gates and the device-specific native gate set.

As an example, let us consider the two QFT circuits for 2 and 5 qubits, respectively. While our algorithm results in a nearly 10-20% reduction in the gate count compared to the `Rigetti` generated circuit, the reduction on the `IBM` platform is just a mere 1.75%, which is due to the difference in the native gate sets, in particular, the availability of *CNOT* as a native gate on `IBM`. Therefore, the *Quantum Fourier Transform* circuits are more resource-efficient when implemented on a `IBM` physical quantum system.

Irrespective of the type of quantum circuit, this evaluation demonstrates that our multi-stage optimization algorithm is universal, i.e., it can easily be ported to other native gate sets by adjusting the gate group aggregation rules, etc.

5.1.2 Quantum circuits with oracles

In contrast to the benchmark circuits from the previous section, the three quantum algorithms considered here involve oracle functions which can be considered a user-definable callback function.

For the benchmarks given in Table 5.3 and Table 5.4, we have considered an abstract oracle function that is safeguarded with barriers to exclude it from optimizations such as gate cancellation and aggregation. In addition, for both, the proposed optimization approach and the state of the art compiler toolchains (`IBM` and `Rigetti`), we have not considered the gates

Table 5.3: Resource evaluation of our approach for oracle-based quantum benchmark circuits generated for the `IBM` platform.

Quantum algorithm	No. of qubits	Optimization level	Size of the quantum algorithm	
			IBM	Ours
Grover	2	1	17 (52.94%)	8
		2	16 (50.00%)	
		3	27 (70.37%)	
Deutsch	2	1	10 (50.00%)	5
		2	10 (50.00%)	
		3	10 (50.00%)	
BV	4	1	22 (50.00%)	11
		2	22 (50.00%)	
		3	22 (50.00%)	

Table 5.4: Resource evaluation of our approach for oracle-based quantum benchmark circuits generated for the `Rigetti` platform.

Quantum algorithm	No. of qubits	Size of the quantum algorithm	
		Rigetti	Ours
Grover	2	15 (46.66%)	8
Deutsch	2	9 (44.44%)	5
BV	4	21 (47.62%)	11

associated with the oracle function in the gate counts reported in Table 5.3 and Table 5.4. As a result, the circuits generated by the proposed and the existing state of the art compilers works for all the oracles possible.

For all three benchmark algorithms, i.e., *Deutsch's algorithm* [50], *Grover's search algorithm* [53] and the *Bernstein–Vazirani algorithm (BV)* [52], our multi-level optimization approach outperforms the `IBM` and `Rigetti` compiler toolchains (roughly reduced by 50%).

5.2 Noise reduction

The above experimental results were obtained by performing the multi-level framework on a classical computer. Using many different modules of `Numpy` [120] and `Pennylane` libraries, the entire multi-level framework was automated. For evaluating the cost function that is required for circuit optimization, a direct *expectation value* function from the `Pennylane` library is used. Therefore, emphasizing the fact that we would get the same results (same count of the number of gates required to implement benchmark circuit) as we would have got by performing the proposed multi-level algorithm as a hybrid mixture of both classical and quantum computation (on an ideal quantum computer). The continuous design optimization is performed using the classical computation where the expectation value can be evaluated (taking the average of the several measurement outcomes) using an ideal quantum computer.

In this section, we will be discussing the implementation of approximate circuits generated using the proposed algorithm and circuits generated using the state of the art `IBM` compiler on real hardware. In particular, we will be focusing on the sparsely connected `ibmq_5_yorktown`

Table 5.5: Average error rate and the execution time associated with all the `ibmq_5_yorktown` supported native gates. In the case of the CNOT(a,b) operation, a is the control qubit whereas b is the target qubit.

Native quantum gates of IBM	Average error rate	Average execution time (10^{-8}s)
R_z	0.000	0.00
SX or $R_x(\frac{\pi}{2})$	0.001	3.55
X or $R_x(\pi)$	0.001	3.55
CNOT(0,1)	0.028	44.08
CNOT(1,0)	0.028	47.64
CNOT(0,2)	0.023	41.95
CNOT(2,0)	0.023	45.51
CNOT(1,2)	0.024	58.31
CNOT(2,1)	0.024	61.86
CNOT(2,3)	0.014	51.91
CNOT(3,2)	0.014	48.35

Table 5.6: Number of physical CNOT operations required to implement the quantum benchmark circuit on the the `ibmq_5_yorktown` device.

Quantum circuit	No. of CNOTs	
	IBM	Ours
Grover's	2	2 (0.00%)
Toffoli	6	4 (33.33%)
Quantum adder	16	12 (25%)
Quantum half adder/peres	7	5 (28.57%)
Quantum inverse peres	7	5 (28.57%)
Quantum nand	6	4 (33.33%)
Fredkin	8	6 (25%)

quantum system.

Figure 5.1 represents the `ibmq_5_yorktown` qubit architecture. In total, it consists of 5 qubits out of which 2 groups of 3 qubits form a complete graph. As a result, a conditional or controlled operation can be performed without any additional swap operation within each group of these 3 qubits. Table 5.5 describes the average error rate and the execution time associated with all the `ibmq_5_yorktown` supported native gates. On an additional note, from Table 5.5, it can be observed that the error rate to perform the *CNOT* operation in this quantum system is quite high compared to the other available elementary native gates.

Figure 5.2 represents a line graph that illustrates the comparison between the 2 qubit *Grover's* circuit generated from the proposed algorithm and the circuit generated by the state of the art `IBM` compiler while implementing it on the `ibmq_5_yorktown` quantum system. In particular, this graph represents the noise/error susceptibility of both the circuits generated. Both the circuits that were generated have been run on the quantum system nearly 20 times (X-axis) with 2048 shots. The Y-axis indicates the probability of measuring the expected outcome where the outcome is obviously the same for both circuits. Both the circuits that

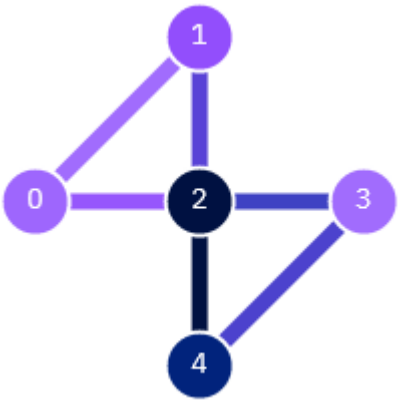


Figure 5.1: `ibmq_5_yorktown` qubit architecture [7].

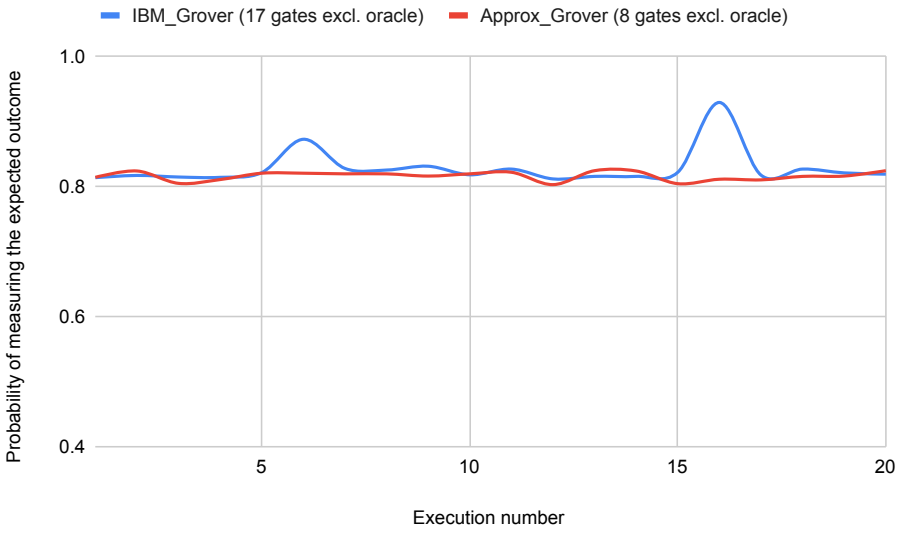


Figure 5.2: Effect of real noise on the 2 qubit *Grover's* circuit.

are being implemented are heavily optimized and the number of gates used is in accordance with what was discussed in the previous section (5.1), i.e., the approximate *Grover's* circuit is implemented using 8 gates (excl. oracle) whereas the `IBM` generated *Grover's* circuit is implemented using 17 gates (excl. oracle). Theoretically, since the approximate *Grover's* circuit required fewer operations compared to the `IBM` generated circuit, we would expect it to be less susceptible to noise. However, from Figure 5.2 it is clear that the approximate *Grover's* circuit does not show any improvement with respect to noise since, apart from the ineffective reduction in the number of $R_z(\theta)$, X and SX gates (less error-prone gates), the $CNOT$ operations required to implement both the circuits are equal in number (Table 5.6).

Now let us consider the *Toffoli* circuit. Figure 5.3 represents a line graph that illustrates the comparison between the *Toffoli* circuit generated from the proposed algorithm and the circuit generated by the state of the art `IBM` compiler while implementing it on the `ibmq_5_yorktown` quantum system. Again, the circuits executed are heavily optimized and

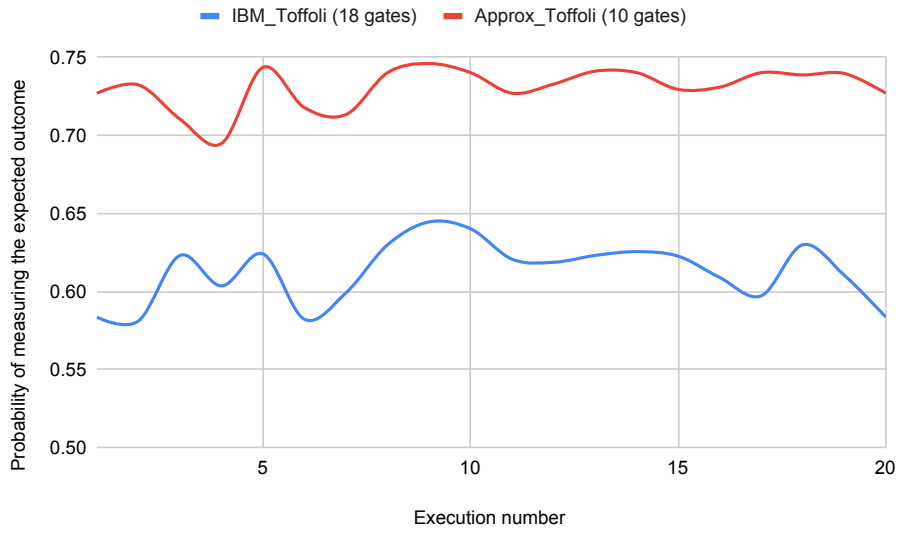


Figure 5.3: Effect of real noise on the *Toffoli* circuit.

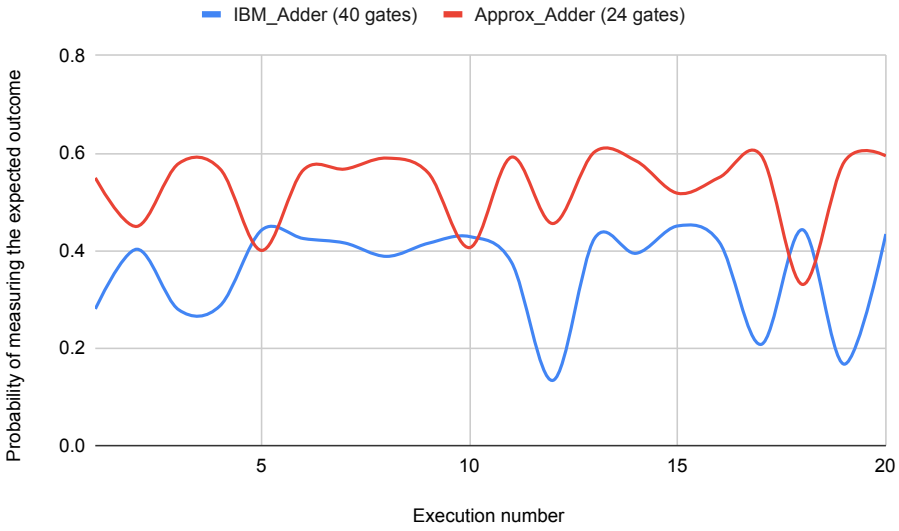


Figure 5.4: Effect of real noise on the 2 qubit *quantum adder* circuit.

the number of gates used is in accordance with what was discussed in the previous section (5.1), i.e., the approximate *Toffoli* circuit is implemented using 10 gates whereas the `IBM` generated *Toffoli* circuit is implemented using 18 gates. From Figure 5.3 it is clear that on a physical quantum device, the approximate *Toffoli* circuit generated from the proposed algorithm is less prone to errors than compared to the `IBM` generated *Toffoli* circuit since the probability of measuring the expected outcome is higher (on average, improved by 20%) for the approximate *Toffoli* circuit. This significant difference is observed due to the reduction in the number of *CNOT* operations (more error-prone gates) (Table 5.6). A similar trend can

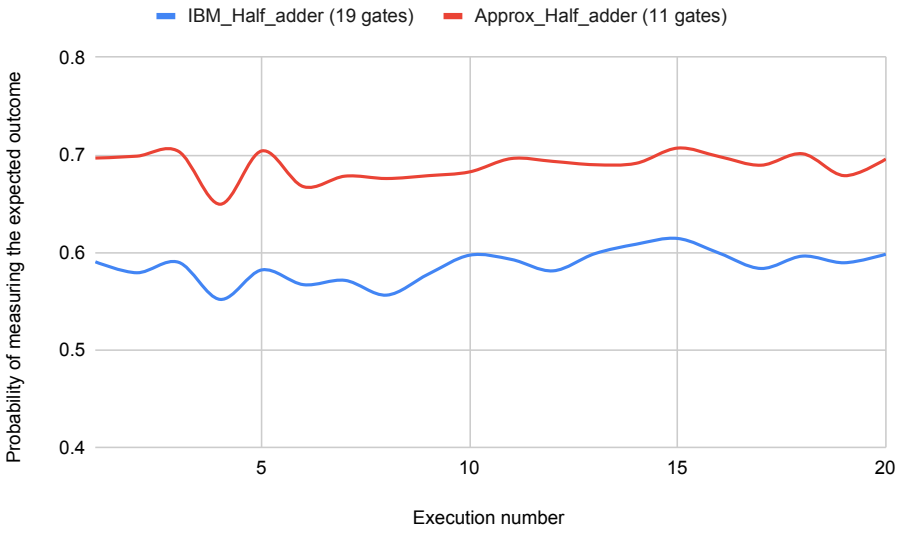


Figure 5.5: Effect of real noise on the *quantum half adder* circuit.

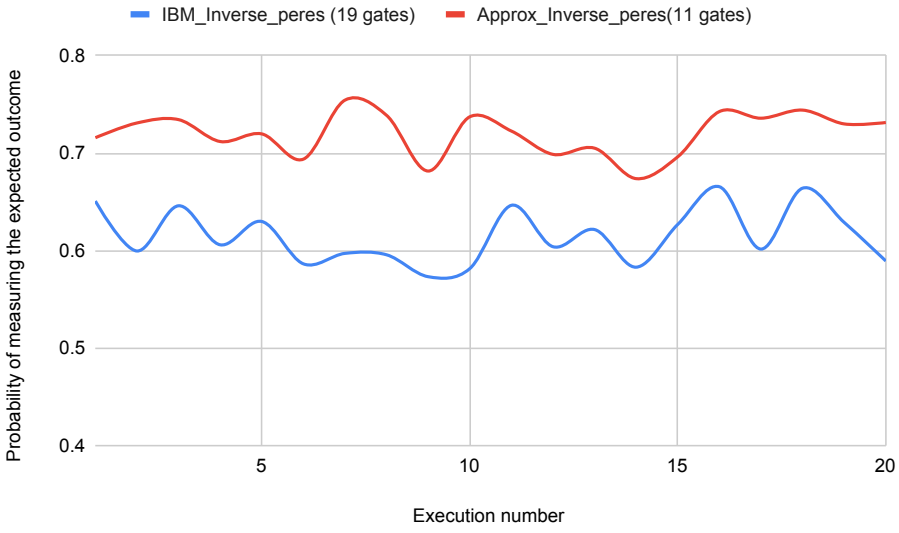


Figure 5.6: Effect of real noise on the *inverse peres* circuit.

be observed in Figure 5.4 which represents the comparison between the 2-bit *quantum adder* circuits (overall 4 qubits) generated by the proposed algorithm and the state of the art `IBM` compiler. Although additional swaps are required for both the circuit generation process, the approximate *quantum adder* circuit is still less susceptible (on average, improved by 47%) to errors when compared to the `IBM` generated *quantum adder* circuit. On an additional note, for both the circuits (*Toffoli* and *quantum adder*), the probability of measuring the expected outcome is not close to 1 since both the circuits implement a high number of *CNOT* operations in their respective circuits.

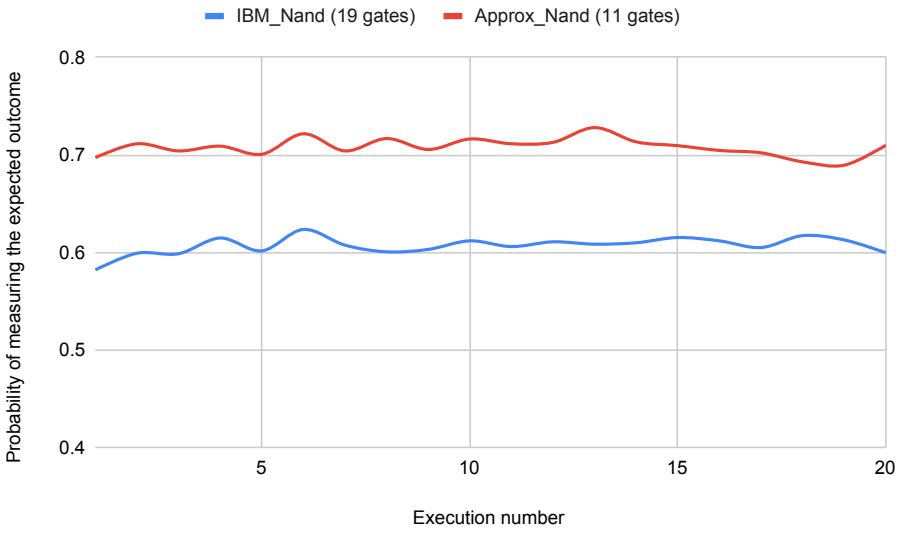


Figure 5.7: Effect of real noise on the *quantum nand* circuit.

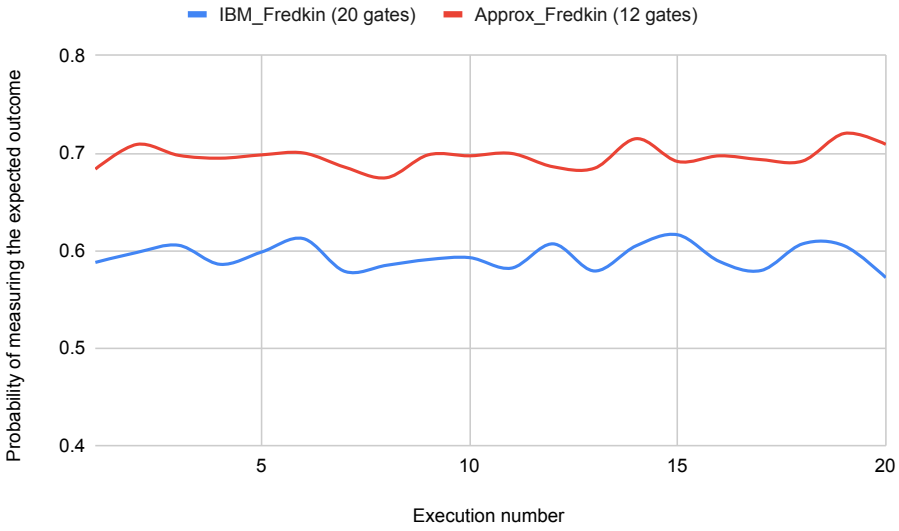


Figure 5.8: Effect of real noise on the *fredkin* circuit.

To emphasize our point, we have considered several other benchmarks which are a mixture of both *CNOT* and *Toffoli* gates. Figure 5.5 represents the comparison between the *quantum half adder* circuits (*peres* gate) generated by proposed algorithm and the state of the art `IBM` compiler. As expected, the approximate *quantum half adder* circuit (*peres* gate) is less susceptible to errors (on average, improved by 19%) when compared to the `IBM` generated *quantum half adder* circuit (*peres* gate) since the number of *CNOT* operations required to implement the approximate circuit is less compared to `IBM` generated circuit (Table 5.6). A similar trend can be observed in Figure 5.6, Figure 5.7 and Figure 5.8 for the circuits *inverse*

peres gate (on average, improved by 18%), *quantum nand gate* (on average, improved by 18%) and *fredkin (controlled-swap) gate (on average, improved by 19%)*, respectively. Also, it is to be noted that we have considered benchmarks that have a unique deterministic outcome (not a superposition of basis states) for every input state since these types of circuits help us in accurately differentiating between the circuits generated from proposed and the state of art compiler of `IBM`. Overall, even from the practical implementation point of view, the notion of " lesser the number of gates ($CNOT$ gates), lesser is the probability of an error occurring " holds good provided that we focus on reducing error-prone native quantum operations.

5.3 Execution time reduction

Table 5.7: Execution time comparison of our approach for quantum benchmark circuits generated for the `IBM` platform.

Quantum circuit	No. of qubits	Execution time (10^{-8}s)	
		IBM	Ours
Toffoli	3	295.82	207.64 (28.80%)
Quantum adder	4	710.39	611.55 (13.91%)
Quantum half adder/peres	3	339.90	251.72 (25.94%)
Quantum inverse peres	3	339.90	251.72 (25.94%)
Quantum nand	3	299.37	211.19 (29.45%)
Fredkin	3	419.54	331.36 (21.01%)

Table 5.7 describes the average execution time of the various benchmark circuits that were considered in Section 5.2. Here we consider the execution time to be the total time taken to implement all the gates associated with any given circuit on a physical quantum system (`ibmq_5_yorktown`). The overhead timings (loading the quantum circuit, validation time, transpilation time, queue duration, etc.) have not been considered since we want to distinguish between the circuits generated using the proposed algorithm and the state of the art `IBM` purely based on gate execution time. In nearly all cases, there is a significant reduction (on average, improved by 24.25%) in the execution time (better performance). This reduction is mainly observed due to the difference between the number of $CNOT$ operations required by the approximate circuit and the `IBM` generated circuit (Table 5.6). In addition, the parallelism between the quantum gates are also taken into consideration for the calculation of the execution time, i.e., if 2 quantum gates are executed in parallel, only the one with the longest execution time is taken into consideration. Overall, performance-wise, the circuits generated using the proposed algorithm are efficiently executed compared to the circuits generated using the state of the art `IBM` compiler.

6 Limitations

In the previous sections, for almost all the quantum circuit benchmarks that were considered, the proposed multi-level optimization approach results in nearly 50% reduction in the size of the circuits when compared to the circuits optimized by the existing state of the art compilers of both `IBM` and `Rigetti`. Moreover, the proposed algorithm not only works for non-oracle based circuits (multi-qubit quantum operations, *Quantum Fourier Transform*, *quantum adders*), it is also applicable for any given quantum algorithm (Deutsch, Grover's and Berstein-Vazirani algorithms) for all its possible oracles circuits. Apart from the fact that it works for all the varieties of circuits, the proposed approach has been tested for circuits that scale up to 16 qubits (*quantum adder*). Therefore, ensuring that the proposed approach can transform any given circuit into a hardware-executable circuit using a minimal number of resources (quantum gates).

But, like every other algorithm, the proposed approach has a few limitations. Although these particular limitations do not alter the functionality of the proposed approach, it is quite significant to be mentioned. Further in this chapter, we will briefly discuss the possible drawbacks to our approach using suitable examples that apply to the `Rigetti` hardware (the same also applies for `IBM`) and also, discuss possible solutions to solve these issues.

Let us consider the *Quantum Fourier Transform (QFT)* circuit for a 2 qubit system as shown in the Figure 6.1. This will be considered as the reference *Quantum Fourier Transform* circuit. Let us also consider the *Quantum Fourier Transform* circuit that is generated from the existing state of the art `Rigetti` compiler and the proposed approach as shown in the Figure 6.2 and Figure 6.3, respectively. Let us assume the input quantum state for all the circuits to be $|00\rangle$. Obviously, for the reference and the circuit generated by the `Rigetti` compiler, the final quantum state after the execution is the same and it is given by the following equation:

$$QFT\,|00\rangle = \frac{|00\rangle + |01\rangle + |10 + |11\rangle\rangle}{2} \tag{6.1}$$

However, for the circuit generated using the proposed multi-level optimization approach, the final quantum state after the execution is given by the following equation:

$$\begin{aligned} Approx\ QFT\,|00\rangle = (0.191 - 0.461i)\,|00\rangle + (-0.461 - 0.191i)\,|01\rangle \\ +(-0.461 + 0.191i)\,|10\rangle + (0.191 + 0.461i)\,|11\rangle \end{aligned} \tag{6.2}$$

The final quantum state generated from the proposed algorithm is not the same as the reference and the `Rigetti` compiled quantum circuit. However, as explained in the previous sections, in particular, the final desired quantum state that is supposed to be achieved by the circuit does not matter as long as the measurement outcomes reflect what is supposed to be achieved. In this case, the circuit generated by our approach behaves similarly to the reference *Quantum Fourier Transform* circuit since, from the hardware perspective, the measurement outcome of the approximate circuit generated is similar to what the reference circuit would

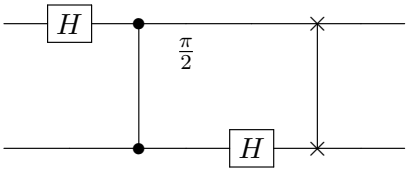


Figure 6.1: Reference *Quantum Fourier Transform* circuit.

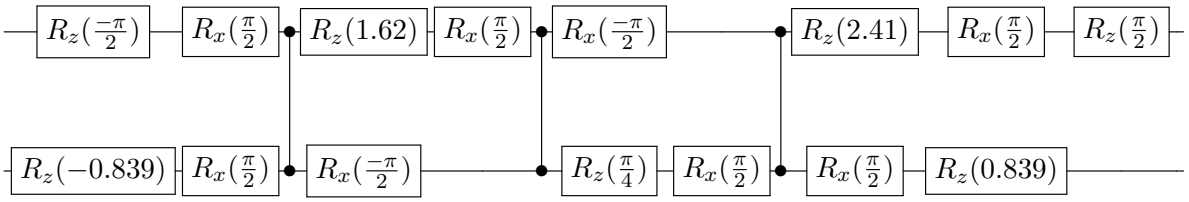


Figure 6.2: `Rigetti` compiled *Quantum Fourier Transform* circuit.

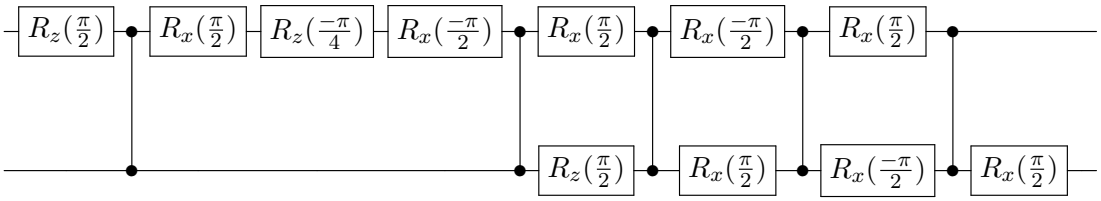


Figure 6.3: Approximate *Quantum Fourier Transform* circuit generated using the propose algorithm.

have resulted in. Based on the above observations, certain conclusions can be drawn and they are as follows:

- From the hardware point of view, the relative and the global phase of the final quantum state achieved through the proposed approach does not affect the measurement outcome as long as the probabilities are similar to the desired final quantum state.
- From the hardware perspective, the amplitudes of the quantum state generated from the proposed algorithm need not be the same as the state generated from the reference circuit as long as the probabilities are similar to the desired final quantum state.

. Keeping these points in mind, let us move on to the next set of observations.

Consider the Figure 6.4 and Figure 6.5 that represents an arbitrary reference quantum circuit " A " and its corresponding approximate circuit " A^* " that is generated using the proposed multi-level optimization approach, respectively. Also, let us assume that the " $+$ " symbol is used to denote the combination of circuit blocks and the symbol " $*$ " represents the optimized approximate circuit that is generated using the proposed approach. From the analyses of the previous sections, it is clear that the optimized circuit " A^* " behaves similar to the reference circuit " A ", i.e., both the circuits have identical measurement outcomes with the same probability. Let us now consider a different scenario. Consider the Figure 6.6 and Figure 6.7 which depicts a combination of arbitrary reference quantum circuits " (A and

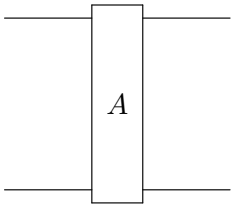


Figure 6.4: Reference circuit A.

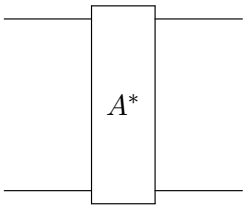


Figure 6.5: Approximate circuit A^* generated using the proposed algorithm.

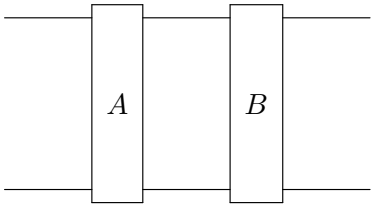


Figure 6.6: Reference circuit $(A + B)$.

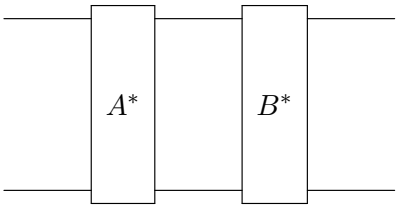
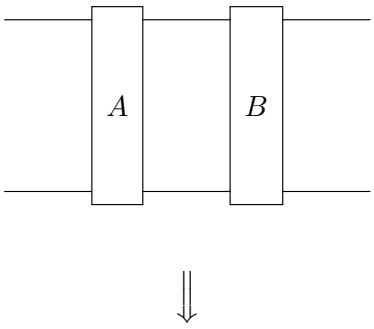


Figure 6.7: Approximate circuit $(A^* + B^*)$ using the proposed algorithm.

B) " and its corresponding combination of individually optimized approximate circuit " (A^* and B^*) " that is generated using our proposed approach, respectively.

Ideally, the optimized circuit " $(A^* + B^*)$ " would behave similarly to the reference circuit " $(A + B)$ ". But, in reality, the approximate circuit " $(A^* + B^*)$ " may or may not have measurement outcomes identical to the reference circuit " $(A + B)$ ". This uncertainty is caused since the circuit " A " and " B " are individually optimized using the proposed multi-level optimization approach. As previously explained, the sole purpose of our multi-level optimization approach is to generate hardware-executable approximate quantum circuits irrespective of the (a) amplitude of the states, (b) relative and the global phases of the state as long as the desired measurement outcomes are achieved. As a result, if the circuit " A and B " are individually optimized using the proposed approach, it is not guaranteed that amplitude, relative and the global phase that was "consumed" while generating the optimized circuit " A^* " will not affect the optimized circuit " B^* " since each circuit is individually optimized and are independent to each other.

(A + B) reference circuit



After passing it through the proposed algorithm

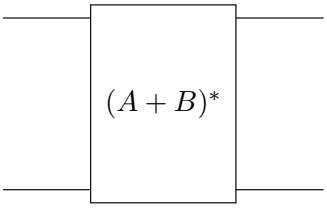


Figure 6.8: Approximate circuit $(A+B)^*$ generated from the reference circuit $(A+B)$.

Therefore, " $A+B \nsim A^*+B^*$ ". But in the case of the existing state of the art compilers, i.e., both the `IBM` and `Rigetti` frameworks, each sub-circuit block can be optimized individually and later combined to achieve the desired final state. Hence, in the case of the existing compiler frameworks, " $A+B \simeq A^{'}+B^{'}$ " where " $A^{'}$ " and " $B^{'}$ " are circuits generated by passing A and B individually through the state of the art compilers.

One possible solution to improve the reliability of the proposed multi-level optimization approach is to optimize the reference circuit as a single entity (single circuit). As a result, if the circuits " A " and " B " are optimized as a single entity, the resulting approximate circuit " $(A+B)^*$ " would perform similar to the reference quantum circuit " $(A+B)$ " as shown in the Figure 6.8. This means that both the circuits " $(A+B)$ " and " $(A+B)^*$ " would have identical measurement outcomes with the same probability. Therefore, " $A+B \simeq (A+B)^*$ ". However, this also raises new challenges. Since the entire circuit should be passed through the multi-level optimizer as a single circuit, there is no room for parallel optimization of the corresponding sub-circuits. As a result, affecting the entire efficiency of the process.

Hence, in general, the proposed multi-stage optimization approach cannot be used to optimize multiple sub-circuits, say, $\hat{U}_1(\boldsymbol{\theta}_{\hat{U}_1})$ and $\hat{U}_2(\boldsymbol{\theta}_{\hat{U}_2})$ individually and pieced them together to a larger circuit as follows $\hat{U}(\boldsymbol{\theta}_{\hat{U}_1}, \boldsymbol{\theta}_{\hat{U}_2}) := \hat{U}_2(\boldsymbol{\theta}_{\hat{U}_2}) \circ \hat{U}_1(\boldsymbol{\theta}_{\hat{U}_1})$. Instead, the so-composed approximate circuit needs to be re-optimized *as a whole* to guarantee correct functioning, whereby the re-optimization process largely benefits from the good initial guess of rotation angles. Also, what *does* work is to prepare the input state with an abstract preparatory circuit that is tacitly assumed to be given in terms of native gates and feed its output as

input into a surrogate circuit that has been optimized stand-alone.

Overall, each approach has its own merits and drawbacks and are highly dependent on the type of circuit that is to be executed. But eventually, the only thing that matters is to implement any given abstract (high level) quantum circuit on a physical device (hardware) by using the least number of possible resources (quantum gates).

7 Conclusion

7.1 Conclusion and findings

In this chapter, the overall conclusion of the thesis is presented. A summary of all the introduced concepts and their corresponding findings is discussed. Also, through this conclusion, the research questions posed in Section 1.3 are briefly addressed.

Overall, the goal of this thesis project was to generate hardware-executable quantum circuits (approximate quantum circuits) using minimal resources (quantum gates). In addition, with the reduction in the number of quantum gates used, we would also expect the reliability and the performance of the executable quantum circuits to be improved. To achieve these objectives, we have considered following an automated approach consisting of different algorithmic components.

Initially, for any given reference/target circuit, the *rule-based approximate gate substitution* technique is applied. This technique forms the baseline structure for generating the executable circuit by replacing gates of higher abstraction level with corresponding approximate gate decompositions. In the case of our thesis, we focus on substituting the *hadamard gates* by approximate *hadamard* gate decompositions (Section 3.1). Although the approach of substitution is similar to the traditional quantum compilers of `IBM` and `Rigetti` where they replace the high-level gates by exact native gate decompositions, the reasoning behind the working of the approximate *hadamard* gate decompositions and the introduction to the concept of quantum gate approximation makes the *rule-based approximate gate substitution* technique a novel approach.

The rotation angles associated with the baseline circuit generated from the *rule-based approximate gate substitution* algorithm are *continuously optimized* until the cost function is reduced to zero or a user-defined tolerance value. The main concept of approximation (approximate computing) is introduced in the *continuous parameter optimization* phase since the resulting parameterized circuit from the *rule-based approximate gate substitution* algorithm is trained to be identical, if not close to the target circuit with respect to the expectation value (Section 3.2). Since the quantum circuit is trained in terms of the expectation value, the resulting circuit is equal to the target quantum circuit only from the hardware point of view (equal with respect to the measurement outcome and not necessarily equal with respect to the final state that is to be achieved). Also, this type of approximation is one of the main reasons for the reduction in the quantum gates required to implement the quantum circuit on a physical platform. The quantum circuit generated from the *continuous parameter optimization* phase is further optimized by the *compiler-based optimization* where quantum gates are either cancelled/negated or fused to reduce the overall count of the quantum gates (Section 3.3). This phase of the multi-stage algorithm has resulted in a significant reduction of the number of quantum gates required for physical implementation.

The concept of the genetic algorithm used in the proposed algorithm (Section 4.2) plays a very important role in carrying out the above mentioned algorithmic components (*rule-based approximate gate substitution*, *continuous parameter optimization* and *compiler-based*

optimization) more efficiently. Since the *rule-based approximate gate substitution* technique is an exhaustive approach, the genetic algorithm employed helps us to significantly reduce the search space. The crossover and the mutation operation also plays a very important role in generating the optimal solution which in our case is the executable quantum circuit with the least number of quantum gates. In addition, the flexibility of the genetic algorithm has the potential to implement our automated approach with a lot more parallelism, therefore, making it computationally even more efficient.

One of the main limitations of the proposed algorithm (Chapter 6) is that it cannot be used to individually optimize parts of the main quantum circuit (cannot be used to individually optimize sub-circuits) instead, the quantum circuit has to be optimized as a whole (as a single entity). As a result, it can become computationally intensive since sub-circuits cannot be parallel and independently optimized.

Theoretically, the reduction in the number of quantum gates should improve the reliability (reduce the susceptibility towards noise) of any given quantum circuit since the probability of the corresponding quantum state (state of the qubit) being altered (due to noise and other environmental factors) also reduces. Turns out that on the `IBM` hardware (`ibmq_5_yorktown`), our proposed approach proves this notion works practically as well (on average, improved by 23.50%). In particular, the proposed approach reduces the usage of *CNOT* operations which significantly reduces the effect of "real noise" since the average error associated with the implementation of the *CNOT* gate is higher compared to the other available native gates. Since the number of resources required is significantly reduced by the proposed algorithm, the average execution time of the corresponding quantum circuits is also reduced (on average, improved by 24.25%).

Overall, the proposed multi-level algorithm has been investigated on different benchmark circuits such as *Quantum Fourier Transform*, *Toffoli*, *quantum adder*, *multi-controlled operations* and three oracle-based algorithms, namely, *Grover's search* algorithm, *Deutsch* and *Bernstein-Vazirani* algorithm. In nearly all cases, even on their highest optimization level, our approach surpasses or is on par with the quantum compiler frameworks by `IBM` and `Rigetti` and produces significantly smaller circuits, up to 50% reduction in the number of gates. We have also considered benchmarks (*peres gate*, *quantum half adder*, *fredkin gate*, etc.) that are a mixture of both, *Toffoli* and *CNOT* gates, map the basis states individually to each other and are feasible to be implemented on the `IBM` hardware (`ibmq_5_yorktown`). The circuits generated by our approach have improved reliability and execution time compared to the circuits generated by the state of the compilers of `IBM` and `Rigetti`. Altogether, the proposed multi-stage algorithm can be viewed as a proof of concept towards the generation of executable quantum circuits.

7.2 Answers to the research questions

In this section, the research questions that were formulated earlier in this thesis (Section 1.3) will be briefly answered.

1. *How the concept of approximate computing is incorporated in the field of quantum circuit generation?*

   (a) *How different is the proposed approximate executable circuit generation technique compared to the existing state of the art executable circuit generation techniques?*

(b) *What are the advantages and disadvantages associated with the proposed circuit generation technique?*

Based on the kinds of literature that were considered in Section 2.4, in the domain of quantum circuit generation, the concept of approximation is introduced by training parameterized quantum circuit either using the expectation value or the unitary matrices as the basis for the cost function. The whole objective behind the concept of circuit generation is to construct a quantum circuit using available resources such that it is identical if not, close to the reference or target quantum circuit [81, 82, 83]. In general, many of the existing approaches use the term approximation due to the mere discrepancies between the trained and the reference circuit unitary matrix. Whereas the proposed approach uses the term approximation by introducing actual gate approximations (Section 3.1), enabling the generation of input specific executable circuits and generating executable circuits with user-defined tolerance (Section 3.2).

One of the main differences between the proposed approximate circuit generation and the existing state of the art circuit generation techniques is the baseline parameterized circuit itself. Most of the existing approaches design the initial baseline parameterized circuits in a way such that they are configured towards an identity matrix. However, there are a lot of circuit permutations that are possible. Since there isn't a fixed baseline structure, this might overall affect the process to find the optimal circuit that we are aiming to achieve. In contrast, our approach starts from the reference circuit and transforms it into an executable one by applying *rule-based approximate gate substitutions* whose parameters are optimized afterwards. This very much simplifies the task of the continuous optimizer since even the initial guess of the so-created circuit inherits some basic structure from the reference one.

Moreover, the proposed algorithm uses the novel concept of rule-based approximate gate substitution technique where the high-level gates of the reference circuit are replaced by parameterized approximate gate decompositions which are later trained to be used as an executable. Whereas the state of the art executable circuit generation techniques (compiler) directly replaces these high-level gates with their respective native gate decompositions. In addition to the above-mentioned difference, unlike the traditional quantum compilers, the proposed approach generates circuits based on user-specific needs, e.g., circuits with tolerance > 0, generate executable circuits that work only for a specific set of necessary input states, etc.

As explained earlier in Chapter 5, the proposed multi-level framework generates hardware-executable approximate circuits which are more

a. resource-efficient (cost-effective),

b. reliable (less susceptible to noise),

c. efficient with respect to execution time.

compared to the circuits generated by the existing state of the art compilers. As mentioned in the limitation Chapter 6, the proposed algorithm currently cannot be used to optimize sub circuits individually since the sub-circuit blocks are dependent on each other. Instead, the circuits should be optimized as a whole using our proposed algorithm. This, therefore, results in compromising the overall computational efficiency of our algorithm. Whereas the state of the art quantum compilers of `IBM` and `Rigetti` allows sub-circuit optimization which can be further used to speed up the process of circuit optimization (if multiple parallel processes are available, then each sub-circuit block can be independently optimized).

2. *How is the proposed circuit generation technique made computationally more efficient using a genetic algorithm? How the flexibility of the genetic algorithm is used to generate a unique set of executable quantum circuits?*

Initially, the proposed algorithm starts with the rule-based approximate gate substitution technique where in our case, we replace all the *hadamard* gates with the approximate *hadamard* gate decompositions (Equations 3.8 – 3.11). As the number of *hadamard* gates increases, the number of possible substitutions also increases exponentially. To overcome this exhaustive search over all the possible substitutions, a genetic algorithm is employed.

The flexibility of the genetic algorithm allows us to visualize every potential approximate quantum circuit in terms of the approximate *hadamard* gate decompositions that are used during the substitution process (integer vector, where every element in the vector is referring to an approximate *hadamard* gate decomposition). As a result of this convenient visualization, genetic algorithm operators like crossover and mutation can be easily performed. With the help of the crossover and the mutation operations, the overall search space is reduced and an optimal set of unique executable approximate circuits are obtained. In addition, the flexibility of the genetic algorithm has to potential to implement our automated approach with a lot more parallelism (optimizations of the potential approximate circuits are independent of each other), therefore, making it computationally even more efficient.

3. *Do the circuits generated from the proposed approach have better reliability compared to the circuits generated from the state of the art quantum compilers? What are the reasons behind these improvements?*

Theoretically, with the reduction in the number of resources (quantum gates), we would expect the approximate circuits generated using the proposed algorithm to be more reliable (less susceptible to noise) compared to the circuits generated by the state of the compilers. However, from the practical implementation point of view, there is no guarantee that the above-mentioned theory would work. As discussed earlier in Section 5.2, it can be observed that there is no improvement (reliability-wise) observed with respect to the 2 qubit *Grovers's* algorithm. Although there is a significant difference (roughly 50% reduction) in the number of elementary native gates used, the overall count of the $CNOT$ operation remains the same for both the circuits (circuits generated by the proposed algorithm and the state of the art quantum compiler of `IBM`). Therefore, to observe a significant improvement with respect to the reliability, either the (less error-prone) the elementary native gates have to reduced by a large extent or (more error-prone) multi-qubit native operations which in our case is the $CNOT$ operation should be reduced to some extent. In the case of the *Toffoli* circuit, we observe a significant reduction in noise susceptibility (on average, improved by 20%) since the number of $CNOT$ operations used by the approximate *Toffoli* circuit is less (4 $CNOT$s) compared to the *Toffoli* circuit generated by the `IBM` compiler (6 $CNOT$s). Overall, from the physical implementation perspective, we could generate a circuit with better reliability provided that we focus on reducing error-prone native quantum operations.

8 Future work

In this chapter, we discuss the potential ideas that can be used in the future to improvise the implementation of the proposed algorithm.

Here are some of the recommended ideas that can be implemented:

- Improve computational efficiency: As explained earlier, the flexibility of the genetic algorithm and independence between the generation of the potential approximate quantum circuits can be used to completely parallelize the entire approach. Although it is stated, it yet hasn't been implemented. Our future work will therefore primarily focus on speeding up the computational workflow by parallelizing the genetic algorithm on parallel computing platforms such as CUDA [121], parallel processing using python [122], etc. and distributed computer systems such as Openstack [123], Hadoop [124], etc.

- Parallel sub-circuit optimization: One of the main limitations of the proposed algorithm is to optimize any given reference quantum circuit as a whole (as a single entity) since sub-circuit optimizations might result in an approximate executable circuit with high inaccuracies. As a result of this lengthy optimization, the computational efficiency is compromised. Potential improvements can be made to the proposed algorithm such that sub-circuit optimizations can be carried out independently. Therefore, enhancing the overall parallelism and computational efficiency of the proposed algorithm.

- Gate approximation: In this thesis, we particularly focused on the implementation of approximate *hadamard* gate decompositions. The concept of *hadamard* gate approximation provided a significant reduction in the number of native quantum operations required to physically implement a quantum circuit. Inspired by the positive results that were achieved, other gate approximations can be analyzed and incorporated in the proposed algorithm for further reduction in the overall quantum gate count of any given quantum circuit.

- Reduction of error-prone gates: As observed in Section 5.2, it is clear that the reduction in the ineffective elementary native gates (less error-prone gates) would not result in any significant circuit reliability improvement. Whereas reduction in the $CNOT$ operations (more error-prone gates) had a significant impact in reducing the noise susceptibility. Therefore, A general strategy for improving the fidelity of approximate circuits is to focus on reducing the number of error-prone gates possibly at the cost of allowing slightly more but less error-prone gates. This as well as other optimization strategies will be investigated in future works.

- Noise adaptation: Throughout this thesis, we have considered generating approximate quantum circuits for an ideal physical quantum device. However, physical quantum systems are affected by noise and other environmental factors that might alter the true state that we are looking for. To further improve the reliability of the quantum system,

the proposed algorithm can be optimized to adapt to different noise environments. In addition, noise models can be integrated into our cost function to potentially mitigate the imperfect behaviour of physical quantum systems.

- Qubit connectivity: In this thesis, we have generated approximate quantum circuits under the assumption of complete qubit connectivity. Therefore, the proposed algorithm can be extended towards different qubit architectures, e.g., gate approximations can be improvised based on the qubit connectivity and other external factors (e.g., average error associated with every qubit can be taken into account).

In addition to the above mentioned potential implementations, conceptually innovative approximation techniques can also be taken into consideration. Different ways of visualizing gate approximations, approximate circuit generation based on different cost functions (unitary distance, etc.), using the genetic algorithm innovatively, etc. are some of the recommendations that can be followed to either improve or provide motivation towards novel approximate circuit generation techniques.

Our future work will therefore primarily focus on speeding up the computational workflow by parallelizing the genetic algorithm on distributed computer systems and on extending the substitution rules and the expressions for the compiler-based optimization sweeps to support a wider range of software-visible gates. This and the above-mentioned ideas are all conceptually supported by our multi-level optimization framework making it a powerful new paradigm for automated quantum circuit generation.

Bibliography

[1] Y.-H. Kim, S. P. Kulik, and Y. Shih, "Quantum teleportation of a polarization state with a complete bell state measurement," *Physical Review Letters*, vol. 86, no. 7, p. 1370, 2001.

[2] L. D. Carlo and D. Elkouss, "Fundamentals of Quantum Information." Presentation, 2020.

[3] M. H. Devoret, A. Wallraff, and J. M. Martinis, "Superconducting qubits: A short review," *arXiv preprint cond-mat/0411174*, 2004.

[4] J. Formosa, "Logic Ciruits." Blog, 2013.

[5] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, M. S. Alam, S. Ahmed, J. M. Arrazola, C. Blank, A. Delgado, S. Jahangiri, K. McKiernan, J. J. Meyer, Z. Niu, A. Száva, and N. Killoran, "PennyLane: Automatic differentiation of hybrid quantum-classical computations," 2020.

[6] J.-G. Liu and L. Wang, "Differentiable learning of quantum circuit born machines," *Physical Review A*, vol. 98, no. 6, p. 062324, 2018.

[7] H. A. et al., "Qiskit: An open-source framework for quantum computing," 2019.

[8] P. Murali, N. M. Linke, M. Martonosi, A. J. Abhari, N. H. Nguyen, and C. H. Alderete, "Full-stack, real-system quantum computer studies: Architectural comparisons and design insights," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, pp. 527–540, IEEE, 2019.

[9] M. A. Nielsen and I. Chuang, "Quantum computation and quantum information," 2002.

[10] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.

[11] D. Castelvecchi, "Ibm's quantum cloud computer goes commercial," *Nature News*, vol. 543, no. 7644, p. 159, 2017.

[12] J. Preskill, "Quantum computing in the nisq era and beyond," *Quantum*, vol. 2, p. 79, 2018.

[13] F. Leymann and J. Barzen, "The bitter truth about gate-based quantum algorithms in the nisq era," *Quantum Science and Technology*, vol. 5, no. 4, p. 044007, 2020.

[14] K. Bertels, A. Sarkar, T. Hubregtsen, M. Serrao, A. A. Mouedenne, A. Yadav, A. Krol, and I. Ashraf, "Quantum computer architecture: Towards full-stack quantum accelerators," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1–6, IEEE, 2020.

[15] P. Selinger, "Towards a quantum programming language," *Mathematical Structures in Computer Science*, vol. 14, no. 4, pp. 527–586, 2004.

[16] D. Zhu, N. M. Linke, M. Benedetti, K. A. Landsman, N. H. Nguyen, C. H. Alderete, A. Perdomo-Ortiz, N. Korda, A. Garfoot, C. Brecque, *et al.*, "Training of quantum circuits on a hybrid quantum computer," *Science advances*, vol. 5, no. 10, p. eaaw9918, 2019.

[17] A. Holmes, S. Johri, G. G. Guerreschi, J. S. Clarke, and A. Y. Matsuura, "Impact of qubit connectivity on quantum algorithm performance," *Quantum Science and Technology*, vol. 5, no. 2, p. 025009, 2020.

[18] R. S. Smith, M. J. Curtis, and W. J. Zeng, "A practical quantum instruction set architecture," 2016.

[19] A. Krol, A. Sarkar, I. Ashraf, Z. Al-Ars, and K. Bertels, "Efficient decomposition of unitary matrices in quantum circuit compilers," *arXiv preprint arXiv:2101.02993*, 2021.

[20] J. C. Bardin, D. Sank, O. Naaman, and E. Jeffrey, "Quantum computing: An introduction for microwave engineers," *IEEE Microwave Magazine*, vol. 21, no. 8, pp. 24–44, 2020.

[21] V. Barkhordarian *et al.*, "Power mosfet basics," *Powerconversion and Intelligent Motion-English Edition*, vol. 22, no. 6, 1996.

[22] J. H. Reina, L. Quiroga, and N. F. Johnson, "Decoherence of quantum registers," *Physical Review A*, vol. 65, no. 3, p. 032326, 2002.

[23] D. Riste, S. Poletto, M.-Z. Huang, A. Bruno, V. Vesterinen, O.-P. Saira, and L. DiCarlo, "Detecting bit-flip errors in a logical qubit using stabilizer measurements," *Nature communications*, vol. 6, no. 1, pp. 1–6, 2015.

[24] L. Tornberg, M. Wallquist, G. Johansson, V. Shumeiko, and G. Wendin, "Implementation of the three-qubit phase-flip error correction code with superconducting qubits," *Physical Review B*, vol. 77, no. 21, p. 214528, 2008.

[25] A. S. Fletcher, P. W. Shor, and M. Z. Win, "Channel-adapted quantum error correction for the amplitude damping channel," *IEEE Transactions on Information Theory*, vol. 54, no. 12, pp. 5705–5718, 2008.

[26] F. Ozaydin, "Phase damping destroys quantum fisher information of w states," *Physics Letters A*, vol. 378, no. 43, pp. 3161–3164, 2014.

[27] C. Cafaro and S. Mancini, "Quantum stabilizer codes for correlated and asymmetric depolarizing errors," *Physical Review A*, vol. 82, no. 1, p. 012306, 2010.

[28] A. W. Cross, L. S. Bishop, S. Sheldon, P. D. Nation, and J. M. Gambetta, "Validating quantum computers using randomized model circuits," *Physical Review A*, vol. 100, no. 3, p. 032328, 2019.

[29] S. Rajan, "Sizing, shape, and topology design optimization of trusses using genetic algorithm," *Journal of structural engineering*, vol. 121, no. 10, pp. 1480–1487, 1995.

[30] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *2013 18th IEEE European Test Symposium (ETS)*, pp. 1–6, IEEE, 2013.

[31] "Qubit." https://en.wikipedia.org/wiki/Qubit.

[32] F. Meier, J. Levy, and D. Loss, "Quantum computing with spin cluster qubits," *Physical review letters*, vol. 90, no. 4, p. 047901, 2003.

[33] T. F. Da Silva, D. Vitoreti, G. Xavier, G. Do Amaral, G. Temporão, and J. Von Der Weid, "Proof-of-principle demonstration of measurement-device-independent quantum key distribution using polarization qubits," *Physical Review A*, vol. 88, no. 5, p. 052303, 2013.

[34] D. Bouwmeester and A. Zeilinger, "The physics of quantum information: basic concepts," in *The physics of quantum information*, pp. 1–14, Springer, 2000.

[35] R. W. Robinett, "Quantum mechanics," *Classical Results, Modern Systems, and Visualized Examples (Oxford: Oxford University Press) ch*, vol. 11, p. 1974, 1997.

[36] R. Horodecki, P. Horodecki, M. Horodecki, and K. Horodecki, "Quantum entanglement," *Reviews of modern physics*, vol. 81, no. 2, p. 865, 2009.

[37] D. Kestenbaum, "Particle decays reveal arrow of time," 1998.

[38] "Quantum entanglement." https://en.wikipedia.org/wiki/Quantum_entanglement.

[39] E. P. Wigner, "Conservation laws in classical and quantum physics," *Progress of Theoretical Physics*, vol. 11, no. 4-5, pp. 437–440, 1954.

[40] A. Furusawa, J. L. Sørensen, S. L. Braunstein, C. A. Fuchs, H. J. Kimble, and E. S. Polzik, "Unconditional quantum teleportation," *science*, vol. 282, no. 5389, pp. 706–709, 1998.

[41] C. Wang, F.-G. Deng, Y.-S. Li, X.-S. Liu, and G. L. Long, "Quantum secure direct communication with high-dimension quantum superdense coding," *Physical Review A*, vol. 71, no. 4, p. 044305, 2005.

[42] J.-W. Pan, D. Bouwmeester, H. Weinfurter, and A. Zeilinger, "Experimental entanglement swapping: entangling photons that never interacted," *Physical review letters*, vol. 80, no. 18, p. 3891, 1998.

[43] D. Aharonov, A. Kitaev, and N. Nisan, "Quantum circuits with mixed states," in *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pp. 20–30, 1998.

[44] I. Glendinning, "The bloch sphere," in *QIA Meeting. Vienna*, 2005.

[45] T. Falter and D. Richter, "Overview of status and challenges of system testing on chip with embedded drams," *Solid-State Electronics*, vol. 44, no. 5, pp. 761–766, 2000.

[46] K. I. Lakhtaria and V. Gadesha, "Fundamentals of quantum computing, quantum supremacy, and quantum machine learning," in *Limitations and Future Applications of Quantum Cryptography*, pp. 21–46, IGI Global, 2021.

[47] F. Jazaeri, A. Beckers, A. Tajalli, and J.-M. Sallese, "A review on quantum computing: From qubits to front-end electronics and cryogenic mosfet physics," in *2019 MIXDES-26th International Conference" Mixed Design of Integrated Circuits and Systems"*, pp. 15–25, IEEE, 2019.

[48] M. Almazrooie, A. Samsudin, R. Abdullah, and K. N. Mutter, "Quantum reversible circuit of aes-128," *Quantum Information Processing*, vol. 17, no. 5, pp. 1–30, 2018.

[49] M. Soeken, R. Wille, O. Keszocze, D. M. Miller, and R. Drechsler, "Embedding of large boolean functions for reversible logic," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 12, no. 4, pp. 1–26, 2015.

[50] D. Deutsch and R. Jozsa, "Rapid solution of problems by quantum computation," *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, vol. 439, no. 1907, pp. 553–558, 1992.

[51] A. Ambainis, "A note on quantum black-box complexity of almost all boolean functions," *Information Processing Letters*, vol. 71, no. 1, pp. 5–7, 1999.

[52] E. Bernstein and U. Vazirani, "Quantum complexity theory," *SIAM Journal on computing*, vol. 26, no. 5, pp. 1411–1473, 1997.

[53] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, (New York, NY, USA), p. 212–219, Association for Computing Machinery, 1996.

[54] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Introduction to linear regression analysis*. John Wiley & Sons, 2021.

[55] R. E. Wright, "Logistic regression.," 1995.

[56] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," *arXiv preprint arXiv:1411.1792*, 2014.

[57] S. Liu, H. Jiang, L. Liu, and J. Han, "Gradient descent using stochastic circuits for efficient training of learning machines," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2530–2541, 2018.

[58] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[59] L. Bottou, "Stochastic gradient descent tricks," in *Neural networks: Tricks of the trade*, pp. 421–436, Springer, 2012.

[60] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," *Cited on*, vol. 14, no. 8, 2012.

[61] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *International conference on machine learning*, pp. 1139–1147, PMLR, 2013.

[62] H. Attouch, Z. Chbani, and H. Riahi, "Rate of convergence of the nesterov accelerated gradient method in the subcritical case α 3," *ESAIM: Control, Optimisation and Calculus of Variations*, vol. 25, p. 2, 2019.

[63] A. Lydia and S. Francis, "Adagrad—an optimizer for stochastic gradient descent," *Int. J. Inf. Comput. Sci.*, vol. 6, no. 5, 2019.

[64] M. C. Mukkamala and M. Hein, "Variants of rmsprop and adagrad with logarithmic regret bounds," in *International Conference on Machine Learning*, pp. 2545–2553, PMLR, 2017.

[65] Z. Zhang, "Improved adam optimizer for deep neural networks," in *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, pp. 1–2, IEEE, 2018.

[66] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, "Quantum machine learning," *Nature*, vol. 549, no. 7671, pp. 195–202, 2017.

[67] C.-C. Chen, S.-Y. Shiau, M.-F. Wu, and Y.-R. Wu, "Hybrid classical-quantum linear solver using noisy intermediate-scale quantum machines," *Scientific reports*, vol. 9, no. 1, pp. 1–12, 2019.

[68] M. W. Johnson, M. H. Amin, S. Gildert, T. Lanting, F. Hamze, N. Dickson, R. Harris, A. J. Berkley, J. Johansson, P. Bunyk, *et al.*, "Quantum annealing with manufactured spins," *Nature*, vol. 473, no. 7346, pp. 194–198, 2011.

[69] P. M. Bakshi and K. T. Mahanthappa, "Expectation value formalism in quantum field theory. i," *Journal of Mathematical Physics*, vol. 4, no. 1, pp. 1–11, 1963.

[70] A. Sampson, J. Nelson, K. Strauss, and L. Ceze, "Approximate storage in solid-state memories," *ACM Transactions on Computer Systems (TOCS)*, vol. 32, no. 3, pp. 1–23, 2014.

[71] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, "Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pp. 258–261, IEEE, 2017.

[72] R. M. Gray and D. L. Neuhoff, "Quantization," *IEEE transactions on information theory*, vol. 44, no. 6, pp. 2325–2383, 1998.

[73] G. G. Guerreschi and A. Y. Matsuura, "Qaoa for max-cut requires hundreds of qubits for quantum speed-up," *Scientific reports*, vol. 9, no. 1, pp. 1–7, 2019.

[74] P. Festa, P. M. Pardalos, M. G. Resende, and C. C. Ribeiro, "Randomized heuristics for the max-cut problem," *Optimization methods and software*, vol. 17, no. 6, pp. 1033–1058, 2002.

[75] D. Coppersmith, D. Gamarnik, M. Hajiaghayi, and G. B. Sorkin, "Random max sat, random max cut, and their phase transitions," *Random Structures & Algorithms*, vol. 24, no. 4, pp. 502–545, 2004.

[76] A. Sarkar, Z. Al-Ars, and K. Bertels, "Quaser: Quantum accelerated de novo dna sequence reconstruction," *Plos one*, vol. 16, no. 4, p. e0249850, 2021.

[77] A. Sarkar, Z. Al-Ars, C. G. Almudever, and K. Bertels, "An algorithm for dna read alignment on quantum accelerators," *arXiv preprint arXiv:1909.05563*, 2019.

[78] K. Mesman, Z. Al-Ars, and M. Möller, "Qpack: Quantum approximate optimization algorithms as universal benchmark for quantum computers," *arXiv preprint arXiv:2103.17193*, 2021.

[79] A. Frisch, H. S. Barowski, M. Brink, and P. H. Roth, "Quantum computing," in *NANO-CHIPS 2030*, pp. 527–548, Springer, 2020.

[80] B. Duan, J. Yuan, Y. Liu, and D. Li, "Quantum algorithm for support matrix machines," *Physical Review A*, vol. 96, no. 3, p. 032301, 2017.

[81] S. Khatri, R. LaRose, A. Poremba, L. Cincio, A. T. Sornborger, and P. J. Coles, "Quantum-assisted quantum compiling," *Quantum*, vol. 3, p. 140, May 2019.

[82] T. Jones and S. C. Benjamin, "Quantum compilation and circuit optimisation via energy dissipation," *arXiv preprint arXiv:1811.03147*, 2018.

[83] M. G. Davis, E. Smith, A. Tudor, K. Sen, I. Siddiqi, and C. Iancu, "Heuristics for quantum compiling with a continuous gate set," *arXiv preprint arXiv:1912.02727*, 2019.

[84] S. Cao, L. Wossnig, B. Vlastakis, P. Leek, and E. Grant, "Cost-function embedding and dataset encoding for machine learning with parametrized quantum circuits," *Physical Review A*, vol. 101, no. 5, p. 052309, 2020.

[85] D. Whitley, "A genetic algorithm tutorial," *Statistics and computing*, vol. 4, no. 2, pp. 65–85, 1994.

[86] G.-x. Zhang, N. Li, W.-d. Jin, and L.-z. Hu, "Novel quantum genetic algorithm and its applications," *Frontiers of Electrical and Electronic Engineering in China*, vol. 1, no. 1, pp. 31–36, 2006.

[87] H. Talbi, A. Draa, and M. Batouche, "A new quantum-inspired genetic algorithm for solving the travelling salesman problem," in *2004 IEEE International Conference on Industrial Technology, 2004. IEEE ICIT'04.*, vol. 3, pp. 1192–1197, IEEE, 2004.

[88] K.-H. Han, K.-H. Park, C.-H. Lee, and J.-H. Kim, "Parallel quantum-inspired genetic algorithm for combinatorial optimization problem," in *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No. 01TH8546)*, vol. 2, pp. 1422–1429, IEEE, 2001.

[89] C. P. Williams and A. G. Gray, "Automated design of quantum circuits," in *NASA International Conference on Quantum Computing and Quantum Communications*, pp. 113–125, Springer, 1998.

[90] M. Lukac and M. Perkowski, "Evolving quantum circuits using genetic algorithm," in *Proceedings 2002 NASA/DoD Conference on Evolvable Hardware*, pp. 177–185, IEEE, 2002.

[91] D. Mukherjee, A. Chakrabarti, and D. Bhattacherjee, "Synthesis of quantum circuits using genetic algorithm," *International Journal of Recent Trends in Engineering*, vol. 2, no. 1, p. 212, 2009.

[92] L. Spector, *Automatic Quantum Computer Programming: a genetic programming approach*, vol. 7. Springer Science & Business Media, 2004.

[93] C. Bravo-Prieto, R. LaRose, M. Cerezo, Y. Subasi, L. Cincio, and P. Coles, "Variational quantum linear solver: A hybrid algorithm for linear systems," *Bulletin of the American Physical Society*, vol. 65, 2020.

[94] N. Wiebe, D. Braun, and S. Lloyd, "Quantum algorithm for data fitting," *Physical review letters*, vol. 109, no. 5, p. 050505, 2012.

[95] M. Benedetti, E. Lloyd, S. Sack, and M. Fiorentini, "Parameterized quantum circuits as machine learning models," *Quantum Science and Technology*, vol. 4, no. 4, p. 043001, 2019.

[96] M. Benedetti, D. Garcia-Pintos, O. Perdomo, V. Leyton-Ortega, Y. Nam, and A. Perdomo-Ortiz, "A generative modeling approach for benchmarking and training shallow quantum circuits," *npj Quantum Information*, vol. 5, no. 1, pp. 1–9, 2019.

[97] P. Li, H. Xiao, F. Shang, X. Tong, X. Li, and M. Cao, "A hybrid quantum-inspired neural networks with sequence inputs," *Neurocomputing*, vol. 117, pp. 81–90, 2013.

[98] M. H. Amin, E. Andriyash, J. Rolfe, B. Kulchytskyy, and R. Melko, "Quantum boltzmann machine," *Physical Review X*, vol. 8, no. 2, p. 021050, 2018.

[99] C. J. Willmott and K. Matsuura, "Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance," *Climate research*, vol. 30, no. 1, pp. 79–82, 2005.

[100] M. Benedetti, D. Garcia-Pintos, O. Perdomo, V. Leyton-Ortega, Y. Nam, and A. Perdomo-Ortiz, "A generative modeling approach for benchmarking and training shallow quantum circuits," *npj Quantum Information*, vol. 5, p. 45, May 2019.

[101] J. C. Garcia-Escartin and P. Chamorro-Posada, "Equivalent quantum circuits," 2011.

[102] G. E. Crooks, "Gates, states, and circuits," 2020.

[103] C. Lomont, "Quantum circuit identities," *arXiv preprint quant-ph/0307111*, 2003.

[104] G. G. Guerreschi and J. Park, "Gate scheduling for quantum algorithms," *ArXiv e-prints*, 2017.

[105] M. Bataille and J.-G. Luque, "Quantum circuits of c – Z and SWAP gates optimization and entanglement." working paper or preprint, Jan. 2019.

[106] M. Ruse, "Charles darwin's theory of evolution: an analysis," *Journal of the History of Biology*, vol. 8, no. 2, pp. 219–241, 1975.

[107] M. Gen, F. Altiparmak, and L. Lin, "A genetic algorithm for two-stage transportation problem using priority-based encoding," *OR spectrum*, vol. 28, no. 3, pp. 337–354, 2006.

[108] A. J. Umbarkar and P. D. Sheth, "Crossover operators in genetic algorithms: a review.," *ICTACT journal on soft computing*, vol. 6, no. 1, 2015.

[109] K. A. De Jong and W. M. Spears, "A formal analysis of the role of multi-point crossover in genetic algorithms," *Annals of mathematics and Artificial intelligence*, vol. 5, no. 1, pp. 1–26, 1992.

[110] E. Semenkin and M. Semenkina, "Self-configuring genetic algorithm with modified uniform crossover operator," in *International Conference in Swarm Intelligence*, pp. 414–421, Springer, 2012.

[111] O. Abdoun, C. Tajani, and J. Abouchabka, "Hybridizing psm and rsm operator for solving np-complete problems: Application to travelling salesman problem," *arXiv preprint arXiv:1203.5028*, 2012.

[112] L. Tang, J. Liu, A. Rong, and Z. Yang, "Modelling and a genetic algorithm solution for the slab stack shuffling problem when implementing steel rolling schedules," *International Journal of Production Research*, vol. 40, no. 7, pp. 1583–1595, 2002.

[113] T.-P. Hong and H.-S. Wang, "A dynamic mutation genetic algorithm," in *1996 IEEE International Conference on Systems, Man and Cybernetics. Information Intelligence and Systems (Cat. No. 96CH35929)*, vol. 3, pp. 2000–2005, IEEE, 1996.

[114] E. Wilson, S. Singh, and F. Mueller, "Just-in-time quantum circuit transpilation reduces noise," in *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pp. 345–355, IEEE, 2020.

[115] J. C. Garcia-Escartin and P. Chamorro-Posada, "A swap gate for qudits," *Quantum information processing*, vol. 12, no. 12, pp. 3625–3631, 2013.

[116] E. Fredkin and T. Toffoli, "Conservative logic," *International Journal of theoretical physics*, vol. 21, no. 3, pp. 219–253, 1982.

[117] M. Möttönen, J. J. Vartiainen, V. Bergholm, and M. M. Salomaa, "Quantum circuits for general multiqubit gates," *Physical review letters*, vol. 93, no. 13, p. 130502, 2004.

[118] L. Ruiz-Perez and J. C. Garcia-Escartin, "Quantum arithmetic with the quantum fourier transform," *Quantum Information Processing*, vol. 16, no. 6, p. 152, 2017.

[119] H. Thapliyal and N. Ranganathan, "Design of efficient reversible logic-based binary and bcd adder circuits," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 9, no. 3, pp. 1–31, 2013.

[120] T. E. Oliphant, *A guide to NumPy*, vol. 1. Trelgol Publishing USA, 2006.

[121] J. Sanders and E. Kandrot, *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010.

[122] L. D. Dalcin, R. R. Paz, P. A. Kler, and A. Cosimo, "Parallel distributed computing using python," *Advances in Water Resources*, vol. 34, no. 9, pp. 1124–1139, 2011.

[123] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "Openstack: toward an open-source solution for cloud computing," *International Journal of Computer Applications*, vol. 55, no. 3, pp. 38–42, 2012.

[124] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*, pp. 1–10, Ieee, 2010.

Appendices

Case Study: CNOT Gate Implementation

A

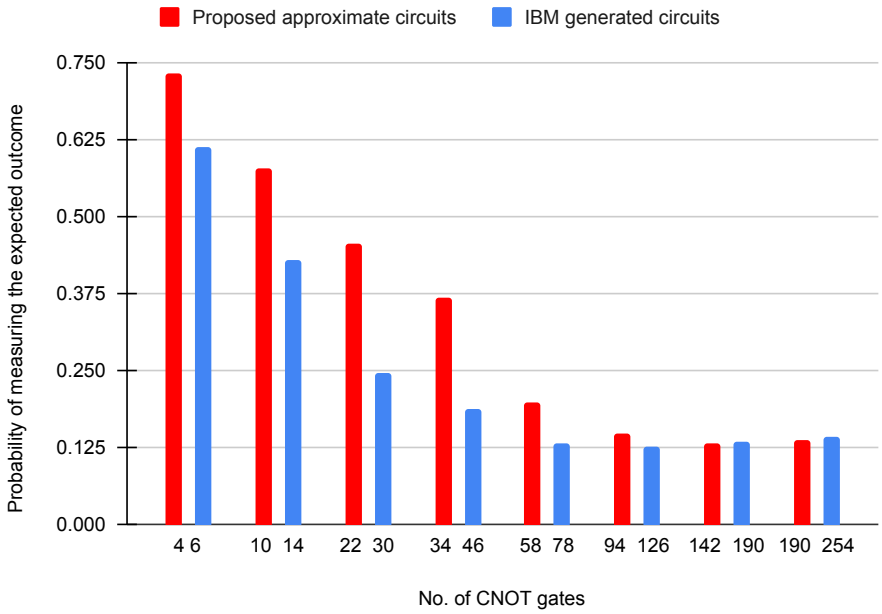


Figure A.1: The effect of real noise as the number of *CNOT* operations increase.

Figure A.1 represents a column graph that illustrates the comparison between the circuits generated by the proposed multi-level optimization approach and the state of the art `IBM` compiler while implementing it on the `ibmq_5_yorktown` quantum system (Figure 5.1). In particular, this graph represents the noise/error susceptibility of both the circuits generated. By circuits, here we are referring to the quantum circuits which are a mixture of *Toffoli* and *CNOT* quantum gates that map any given input quantum state (individual quantum state, not a superposition of quantum states) to a deterministic output quantum state. On an additional note, all the circuits considered for this case study are of 3 qubits and are constructed by extending a particular circuit template consisting of only *Toffolis* and *CNOTs* quantum operations (which are eventually decomposed into *CNOTs* and other elementary native gate operations). All the circuits that were generated have been run on the quantum system nearly 20 times with 2048 shots. The Y-axis indicates the average probability of measuring the expected outcome where the outcome is obviously the same for the corresponding circuit pairs. The X-axis represents the number of *CNOT* operations corresponding to the circuits generated by the proposed algorithm and the state of the art `IBM` compiler.

As discussed earlier (Section 5.2), from Table 5.5, it can be observed that the error rate to perform the *CNOT* operation in this quantum system is quite high compared to the other available elementary native gates. Therefore, as the number of *CNOT* operations increase

(as we move along the X-axis), the probability of measuring the correct outcome starts to proportionately decrease (on average, decreased by 21% for the approximate circuits generated by the proposed algorithm and on average, decreased by 32% for the circuits generated by the `IBM` compiler). This trend can be clearly observed for the circuits that are presented in Figure A.1. Also, as the number of *CNOT* operations increase (as we move along the X-axis), eventually, both, the circuits generated using the proposed approach and the circuits generated using the `IBM` compiler measure the expected outcome with nearly the same probability, therefore, indicating that after a certain number of *CNOT* operations or error-prone operations (which in our case is the circuit with 94 *CNOT* operations) both, the circuits generated using the proposed algorithm and the state of the art `IBM` compiler reach a point of saturation (completely mixed quantum state) with respect to the probability of measuring the expected outcome (Y-axis). Also, since the number of *CNOT* operations required to implement the approximate circuits is less compared to the `IBM` generated circuits, from Figure A.1 it is clear that on a physical quantum device, the approximate circuits generated from the proposed algorithm is less prone to errors than compared to the `IBM` generated circuits since the probability of measuring the expected outcome is higher (on average, improved by 57%) for the approximate circuits.

In addition to the above-discussed points, another important observation can be made. Since all the circuits considered are of 3 qubits, the state generated by the circuits can be in the form of a completely mixed state [22, 43] if the probability of measuring the expected outcome is equal to $\frac{1}{2^3}$, i.e., there is an equal probability of measuring all the 8 possible states. In Figure A.1, the circuit with 78 *CNOT* operations that is generate by the `IBM` compiler has nearly reached the completely mixed state whereas, the circuit with 94 *CNOT* operations is still well above the mixed state threshold (above 0.125). Although the number of *CNOT* operations are more (94), the probability of measuring the expected outcome is better compared to the circuit with relatively less *CNOT* operations (78). Therefore, from the physical implementation point of view, apart from the effect of the error-prone quantum gates, there are a lot of other factors (input quantum state, transition between the input and the output quantum state, etc.) that overall affect the probability of measuring the expected outcome. This unusual behaviour will be further investigated in future works.