Machine-Learning driven Analysis and Optimization of Quantum Circuits for Noisy Near-term hardware

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science
in
Computational Natural Sciences
by Research

by

Animesh Sinha
2018113001
animeshsinha.1309@gmail.com

International Institute of Information Technology
Hyderabad - 500 032, INDIA
March, 2022

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "Quantum Circuit Optimizations using Reinforcement Learning" by Animesh Sinha, has been carried out under my supervision and is not submitted elsewhere for a degree.

April 12, 2022

Adviser: Prof. Harjinder Singh

To Compute

Acknowledgments

I want to express my gratitude to my advisor, Prof. Harjinder Singh, for helping me take my first steps in research, help explore the many domains I might be interested in, and eventually helping me start making progress on Quantum Computing. I would also like to thank my senior, Utkarsh Azad, for mentoring me through the entire process, from formulating the problem to benchmarking the results on both the projects and pricing the requisite domain knowledge and intuition required to tackle those problems. Prof. Harjinder Singh and Utkarsh Azad have also helped write and edit this thesis immensely.

Next, I would like to thank the professors in the CCNSB and CQST departments for designing courses that helped me gain interest in the sciences, specifically Quantum. Prof. Subhadip Mitra, Prof. Deva Priyakumar, Prof. Santanav Chakraborty, and many others have taken courses that have helped hone my skills in this domain.

I would also like to thank my friends who helped with techniques learned and ideas developed in the project. Bhuvanesh Sridharan helped out a lot with the Monte Carlo Tree Search formulation of the problem and helped find ways to improve the project iteratively. Kalp Shah and Jai Bardhan contributed valuable thoughts on the scientific and machine learning aspects of the projects involved. Other friends and lab-mates, including Adrian Alva, Shweta Sahoo, Gaurang Tandon, Kanish Anand, and many others, have engaged in discussions on these topics, which have been invaluable to me in validating and extending my ideas.

Finally, I would like to thank my Mom and Dad for their constant support and guidance on all matters, including academic and technical advice.

Abstract

Quantum Computation is one of the most promising computing and scientific simulations futures. However, the present-day quantum devices suffer from several limitations, a small number of qubits, limited connectivity, noisy evolution, and others. Therefore, the need of the hour is to come up with both hardware-based and algorithmic changes to mitigate these limitations and put forth a step toward a quantum computer that achieves supremacy over its classical counterpart. The primary focus of this dissertation is to present a method of efficiently compiling Quantum Circuits on present-day hardware to minimize the effects of limited connectivity and effects of noise. Further, we explore a class of hybrid quantum-classical algorithms called variational quantum circuits and attempt to characterize their properties, evolution, and advantages.

First, we provide the requisite background in Quantum Computing, Variational Quantum Methods, Deep Learning, and Reinforcement Learning. Next, we present qRoute, a Reinforcement Learning based solution for compiling Quantum Circuits onto present-day hardware. We elucidate the method that QRoute uses for depth minimized compilation, which is essentially a Monte-Carlo Tree Search put together with a Graph Neural Network to decide which parts of the tree to explore. We discuss the details of the algorithm, the key points of innovation that differentiate it from the previous methods, and the state-of-the-art results it achieves on various circuits compilation benchmarks. Finally, we move to the application of quantum computers in solving real-world problems and discuss the circuits called Variational Quantum Circuits. We present a framework qLEET for characterizing the training paths, loss landscapes, entanglement capability, and expressibility of these circuits, and we provide a use-case example in analyzing an algorithm called QAOA for computing the max-cut of a graph, which is an NP-complete problem and require time exponential in the number of nodes on a classical computer. All code for qRoute and qLEET has been released in the open-source community and provides easy and modular access to endpoints where our algorithms can be tweaked for further research in this domain.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Scope of the Thesis

The present-day noisy intermediate-scale quantum computers are capable of running simple quantum procedures, and in the case of some particular problems, they come close to showing a significant quantum advantage over their classical counterparts [4]. However, we are still a long way off from the goal of performing general-purpose computation to solve significant problems with a substantial speedup over the classical realm. Making quantum computation feasible will involve iterated progress in several domains, like the following.

1. Building hardware with a larger number of qubits that have better noise-resilience and higher connectivity for multi-qubit operations across those qubits.

2. Designing quantum error detection codes to mitigate noise by composing a single logical qubit from many physical qubits.

3. Coming up with quantum algorithms to solve problems of practical value which are intractable on classical computers.

4. Compiling those algorithms down to circuit operations such that they can be carried out quickly and reliably.

This focus of the work in this dissertation is to address points 3 and 4.

1.1.1 Research Problems tackled

We need to tackle the following computational challenges to address the problems listed above. This thesis proposes novel solutions to all of the problems listed below, outperforming the state-of-the-art solutions on large classes of Quantum Circuits.

T1 *To incorporate the notion of parallelizability and noise mitigation in Quantum Circuits in Machine Learning based circuit routing algorithms* The longer quantum circuits take to execute, the more

noise and decoherence of the quantum state affect the final results. Quantum states decohere even quicker when no operations are being applied to them, which is when they wait for parts of the circuit to finish. Planning methods to execute circuits with the least number of gates do exist, but we need to add the notion of parallel operations into this planning process and any neural process that helps guide it.

T2 *To design an algorithm for efficient and neurally-guided search in combinatorially large search spaces.* A parallelizable set of actions need to be scheduled at each timestep by our planner. The number of possible sets of operations we are deciding over is exponential in the hardware size. Since searching overall sets is infeasible, all methods to iteratively add or remove elements from the set to develop some heuristic maximization. We attempt to develop one such method that would allow us to train our networks in this RL setting stably.

T3 *To develop a framework for analyzing variational algorithms on noisy-quantum computers, evaluating the quantum advantage, convergence properties of the learning process, etc.* Variational Methods 2.2.2 are typically used to solve hard optimization problems, in which the classical subsystem learns parameters for a Parametrized Quantum Circuit (PQC) to maximize some function of the state prepared by said circuit. This is, in essence, a learning algorithm, and analysis of these learning algorithms and iterating on designs of these circuits should be both based on intuition from other quantum algorithms (in the way QAOA is inspired by quantum annealing and trotterization 2.2.2.1) and from data obtained about the loss landscape on which we are optimizing.

1.2 Motivation

This section discusses the specific features of the problems discussed in this dissertation that motivate elements of our method. We enlist other approaches that bear similar motivation but fail to achieve results, together with some explanation of this failure.

1.2.1 qRoute: Qubit Routing

The design and development of quantum algorithms are often hardware-agnostic fashion, i.e., the assumption is that any two qubits can be operated on, and all qubits have perfect fidelity. However, since it is physically infeasible to allow multi-qubit operations between arbitrary pairs of qubits, we need to decompose existing gates into sets of SWAP gates to generate physical proximity for operations to be feasible, as shown in Figure 1.1.

While solving the problem of routing, we note that the value of the entire set of actions scheduled at each timestep is mainly dependent on the independent value of actions in the set and the values added by the co-occurrence of small subsets of these actions, i.e., operations occurring on qubits significantly separated on hardware do not affect each other. This strikes a remarkable similarity with the general

2
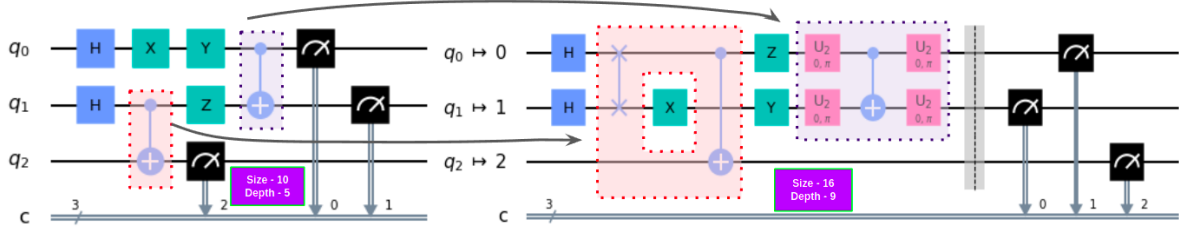
Figure 1.1: Transformation of a Quantum Circuit where non-local operations are scheduled to one implementable on the hardware (qubits 1 and 2 are not a local pair, 0 and 1, and 0 and 2 are). The decomposition of some gates is shown with arrows.

problem of search in combinatorial spaces [18], an improvement in the solution to the routing problem with implying a new method to decide over subsets, as will an improvement in a decision over subsets most likely imply an improvement in our router.

Annealers with Dense neural networks as proposed in [18, 38] were the only previous works that solve qubit routing phrased as a decision over sets problems. We noted the following challenges while running replications of both these results:

1. Backpropagating rewards across the annealers is noisy; the combining algorithm needs to be a more deterministic function of its inputs.

2. Value function is hard to estimate, particularly in large circuits, since it requires tremendous foresight of how long it will take to schedule the rest of the circuit, while optimal actions can be chosen from a very shallow look ahead.

3. Annealers are slow in high-value set construction and are infeasibly slow with a large number of qubits or gates.

4. The simulations need to be run on very long event horizons; however, most of the actions should affect rewards in the near future.

5. The value function should model the symmetries inherent to the system and understand that swap actions chosen are only dependent on the local structure around the qubit.

Over the past few years, reinforcement learning has shown remarkable breakthroughs in playing games like Go[45], Chess[47] using planning with Monte Carlo Tree Search, playing StarCraft[56] and Dota[35] using an actor-critic method with combinatorial decisions for all agents taken using an actor-critic method trained in an autoregressive setting. These games are faced with tasks similar to those listed above, with long horizons to reason over, combinatorial search spaces requiring the coordination of many controlled agents, etc., being major issues. The fact that the methods used to solve these games can cope with these challenges motivated us to try them on the problem of routing as well.

Challenge 1 is well handled by using MCTS instead of the annealer setting. A value network and a policy network are used, which helps neurally prune the tree helping with challenge 3, and the ability to model the policy directly in the policy network and to train it in an actor-critic setting to solve challenge 2. Greedy gate scheduling and intermediate rewards on each step as gates are scheduled to help avoid long-horizon reasoning, addressing challenge 4. The Graph Neural Network with a small number of message passing rounds can model symmetries and localities well solving 5.

Following the publication of our work, Zhou et al. [63] have proposed a similar MCTS-based method (without a neural guide) for depth minimization. This method shows promising results amongst other classical planners and heuristic methods and shows that the choice of setting is valid even independent of the Machine Learning problem. Our method, however, is capable of routing larger circuits with better speed and lower depth.

1.2.2 qLEET: Variational Circuit Property Visualizations

Variational Methods (discussed in section 2.2.2) approach optimization problems, i.e., maximization or minimization under constraints, by posing them as the task of finding the ground state eigenvector of a hamiltonian that models the problem. A parametrized quantum circuit generates the ground state with guessed parameters and the guess by a classical system.

Depending on the algorithm we use, the problem Hamiltonian as a function of the parameters of our prepared state can be easy or hard to optimize over. In classical machine learning, we have seen that model architecture, which is essentially the way the model is parametrized, can affect the loss landscape it is optimizing over, and models which result in smooth landscapes perform well, and those which do not show inferior results. The most striking of these demonstrations is by [24], as demonstrated in the image 1.2.

(a) Loss Landscape without Skip Connections   (b) Loss Landscape with Skip Connections
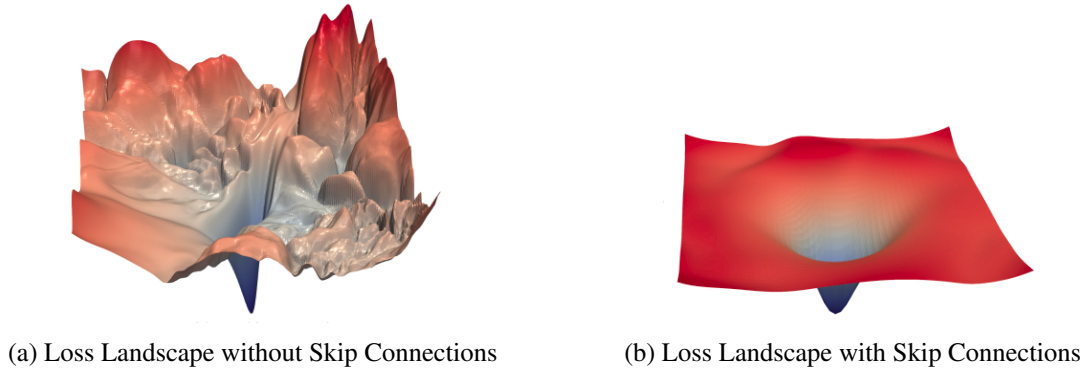
Figure 1.2: The loss landscapes of neural networks with and without skip connections, as visualized by Li et. al.[24]. The architectural decision of adding skip connections makes the feasibility of the optimization process a lot higher, to the extent visualizable on a 2-D random projection.

If the landscape we are optimizing over is not smooth, the optimization process is easily affected by choice of initial variables. Furthermore, the convergence of differently initialized optimization runs,

or lack thereof, tells us about the degree of abundance of local minima on this landscape. One notable example of such analysis is by Lorch [26]. We hope that by using a similar analysis on parameterized quantum circuits, we can compute which class of circuits leads to better optimizability over the underlying hamiltonian. We give an example analysis of this method on circuits like QAOA for computing the Max-Cut of a graph (formulation discussed in 2.2.2.1) amongst others.

Furthermore, given that we are leveraging quantum computers to solve these problems and optimize over the loss landscape, we also need to ensure that these quantum resources are being efficiently utilized. For this, Sim et al. [49] have come up with two metrics, expressibility and entanglement capability.

- The parametrization of the Variational Circuits affects what set of states it can generate; there can always exist some set of states which are not generatable by our parametrized ansatz for any choice of the parameter-vector. The amount of the state space that our circuit can generate is measured using the metric of expressibility; the higher the expressibility of the circuit, the better it is for exploring the entire space of solutions.

- The quantum speedup in these variational circuits is attributed to entanglement between the qubits (discussed in section 2.1.2), due to which the information being processed inside of the quantum circuit is exponentially greater than that which would have been by a classical system with the same number of qubits. A measure of how much of this entanglement capability is used by our PQC is a valuable metric; its presence seems essential to obtaining an advantage over any classical method.

In our PQC analysis software qLEET, build a backend agnostic suite of python packages that help evaluate this metric for well-known variational algorithms like QAOA for Max-Cut on a Graph, Variational Quantum Eigensolvers, and the like. We include support for doing these evaluations on ideal conditions with pure state vectors or with density matrices and noise models. The final evaluation can be computed by sampling the quantum state or mathematically evaluating these metrics by directly accessing the underlying quantum state to speed up the computation.

1.3   Thesis Layout

C1 This is the introductory chapter, which discusses the scope of the work carried out in this thesis in the context of developments in quantum computation, addresses the problems we are attempting to pose solutions to and adds some motivation for the methods that we will develop in the following chapters.

C2 Here, we present a background in Quantum Computing and Reinforcement Learning which is requisite for understanding the motivations of the methods developed and the algorithms used in the remainder of this dissertation. We conclude this chapter by enlisting some ideas that we are going to use to address the problems posed in 1.1.1.

C3   As the first major contribution of this thesis, we present **qRoute: Qubit Routing using Graph Neural Network aided Monte Carlo Tree Search**, which is a reinforcement learning algorithm we propose for depth-minimized (used as a proxy for noise-mitigated) compilation. We discuss the routing problem and the specifics of our algorithm and associated neural architecture design.

C4   The other contribution of this thesis is **qLEET: Visualizing Loss Landscapes, Expressibility, Entangling power and Training Trajectories for Parameterized Quantum Circuits**, in which we present a way to analyze the properties of variational methods that can be implemented on present-day quantum computers, and build a software framework for the same.

C5   We conclude with a summary of methods and results discussed in this thesis, the practical use-cases of these methods, and the scope of extension of this work in the future.

Chapter 2

Background in Quantum Computation and Reinforcement Learning

This chapter introduces the basics of quantum computing and reinforcement learning that form the backdrop of the work in the remainder of the thesis.

We start with a brief introduction of the quantum computation model and its mathematical and geometric formalism and introduce the circuit model of computation.

Next, we start a discussion on Quantum Algorithms, starting with pure quantum algorithms like Grover's, Shor's, and Hamiltonian simulation and moving on to Variational Methods like QAOA (Quantum Approximate Optimization Algorithm) and VQE (Variational Quantum Eigensolver). The compilation and visualization of these algorithms form the basis of the work presented in this thesis. Both our algorithms have been tested extensively on some subset of these circuits. There are a small number of such subroutines that have currently been discovered in quantum algorithmic theory, and these subroutines are often and repeatedly used, motivating the value of performing routing as close to optimally as possible on these circuits (Routing results on these algorithmic benchmarks as setup by IBM are listed in A). The QAOA algorithm for max-cut is also our test-bed for analyzing and visualizing the learning properties of these variational circuits.

Finally, we move to a didactic introduction to reinforcement learning the different settings like Value-based, Policy-based, etc. We add some mathematical/algorithmic details to those settings and algorithms relevant to our work, like Actor-Critic methods, Monte Carlo Tree Search, etc.

2.1   Quantum Computation

2.1.1   Qubits and Quantum Computation Model

Quantum Computers store information as quantum bits, or qubits. These qubits can be evolved by operating on them with unitary operators, also called gates. The gate model of computation is rather familiar; the following is a diagrammatic illustration of the same:
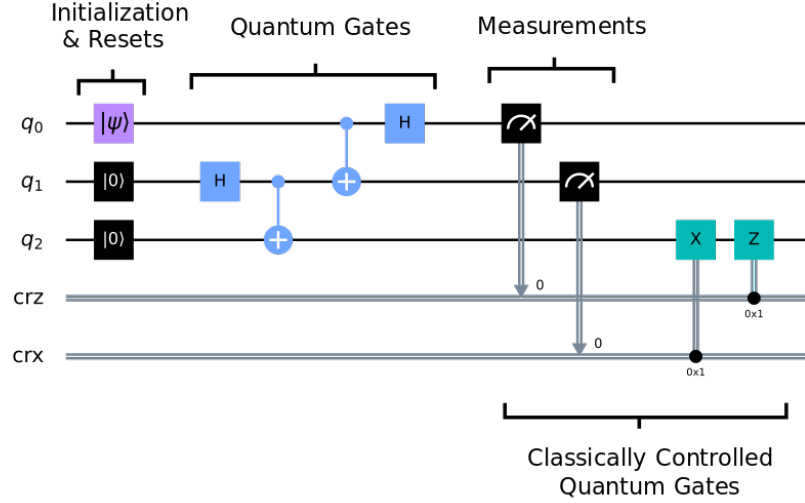
Figure 2.1: The image shows the parts of a typical quantum circuit, with 3 qubits represented by the wires and a set of gates applied to them, followed by measurement of those qubits.

A classical bit can be either 0 or 1. However, a qubit can live in any state inbetween 0 or 1, which is understood as being in a weighted superposition of the 0 and 1 states. So the state of a qubit

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \quad \text{such that } \alpha^2 + \beta^2 = 1 \text{ and } \alpha, \beta \in \mathbb{C} \tag{2.1}$$

where the normalization of probabilities forces. However this state of the qubit is not accessible to us, and we can only measure the qubit probabilistically, with probability of being $|0\rangle$ being α^2 and that of $|1\rangle$ being β^2.

Each qubit, in addition to the superposition it is in, also has a phase term, which is represented on the Bloch-sphere 2.2 on the x-y plane. The phase does not affect the immediate measurement of the qubit but can affect the resultant phase and superposition when some unitary operation is applied to the qubit.

2.1.2 Unitaries, Gates, and Entanglement

The state of a system of qubits can be modified by applying unitary gates on those qubits.

Some multi-qubit gates introduce a property called Entanglement. Examples of such gates are CNOT, CZ, etc. Once two qubits are "entangled" through some such operation like CNOT, their state cannot be written as a product of the individual states of the qubits; they are now linked in a way that the probability distribution of collapsing either state along the measurement axis is not factorizable.

8
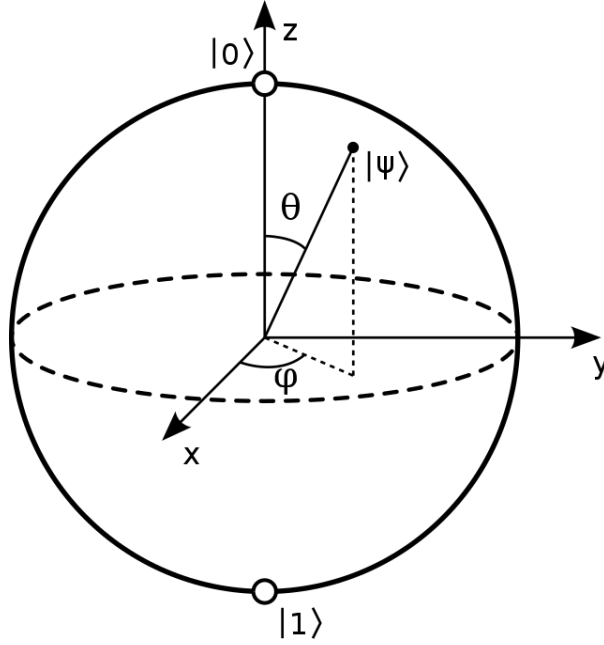
Figure 2.2: Bloch Sphere represents the state of a qubit $|\psi\rangle$. The pure states $|0\rangle$ and $|1\rangle$ are the vectors along the z-axis on the opposite poles. The angle along the x-y plane represents the phase of the qubits.

2.1.3 Density Matrices and Noise

State preparation on a quantum computer is subject to noise. Therefore there often exists some uncertainty in the state of qubits on actual physical hardware. This uncertainty can be modeled as a classical probability distribution over the different quantum states that might have been prepared. Such a state is called a mixed state, as opposed to a pure state. Mixed states can be represented using Density matrices ρ, which are 2-D matrices with $2^n \times 2^n$ elements for n qubits.

$$\rho = \sum_i p_i |x_i\rangle \langle x_i| \tag{2.2}$$

Much like unitary operations on state vectors, the application of unitaries on density matrices can be represented through simple matrix multiplication:

$$\mathcal{O}(\rho) = \sum_i p_i \mathcal{O}(|x_i\rangle)\mathcal{O}(\langle x_i|) = \sum_i p_i U |x_i\rangle \langle x_i| U^\dagger = U\rho U^\dagger \tag{2.3}$$

2.2 Quantum Algorithms

2.2.1 Purely Quantum Algorithms

Superposition and Entanglement together provide quantum computers with natural parallel processing power. While full access to this parallelism gets bottlenecked at the measurement layer since we can

9

Operator	Gate(s)		Matrix

Pauli-X (X) — $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

Pauli-Y (Y) — $\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$

Pauli-Z (Z) — $\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

Hadamard (H) — $\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$

Phase (S, P) — $\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$

$\pi/8$ (T) — $\begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$

Controlled Not (CNOT, CX) — $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$

Controlled Z (CZ) — $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$

SWAP — $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Toffoli (CCNOT, CCX, TOFF) — $\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$

Figure 2.3: Popular logic gates along with their in-circuit representations and corresponding unitary matrices.

sample only one of the many states in weighted superposition, it is conceivable that for many an algorithm, this parallelism can result in a processing speed-up. A near-term goal with Quantum Computers is to achieve quantum supremacy, which is to solve a problem (possibly one of no practical use) that no classical computer can solve in a feasible amount of time. [54]

- Grover Like (Amplitude Amplification class of algorithms).

- Shor's Like (Using Quantum Phase Estimation and Quantum Fourier Transform)

- Hamiltonian Simulation

2.2.1.1 Grover's Search

Grover's search is an algorithm to search in an unordered list of size n in $\sqrt{n}$ time proposed by Grover in 1996 [16].

The Problem : Grover attacks the problem of unstructured search, where we have a list of n elements in any permutation, and we have an oracle that marks each of these elements 1, which is one of the results of the search procedure we wished to find, or as 0 which is for those elements which were not one of those elements. We are told that the oracle only returns a value of 1 for some m of those elements, where $m << n$.

Overview of the Algorithm : Following is a brief explanation of how the Grover's Algorithm operates:

1. **Preparing the initial superposition of bitstrings:** The initial state should be the superposition of all elements in our search domain. Since these are the indices of the elements we are searching over, we can take this to be an equal superposition of all basis states, constructed by applying the hadamard gate over all qubits.

$$|x\rangle = |s\rangle = \frac{1}{\sqrt{n}} \sum_{i=1}^{n} |b_i\rangle \tag{2.4}$$

2. **Application of a phase-kickback oracle:** For any input state $|x\rangle$, if it is a valid solution (i.e. $f(x) = 1$), then the oracle that bitstring with a negative phase. If $|x\rangle$ is not a basis state but rather is a superposition of states, then the oracle operates on each basis component of the state independently as shown in equation. 2.5

$$\mathcal{O}\big(|x\rangle\big) = \mathcal{O}\bigg(\sum_{i} w_i(x)\,|b_i\rangle\bigg) = \sum_{i} w_i(x) \begin{cases} -\,|b_i\rangle & \text{if } f(b_i) = 1 \\ |b_i\rangle & \text{if } f(b_i) = 0 \end{cases} \tag{2.5}$$

3. **Performing a reflection around the average amplitude:** Following the application of the oracle, we can apply a reflection around the mean amplitude of the superposition of all solutions via an application of the Diffuser operator.

$$\mathcal{D}\big(|x\rangle\big) = \big(2\,|s\rangle\,\langle s| - 1\big)\,|x\rangle \tag{2.6}$$

Given that there are small number of solutions

4. **Repeat 2 steps above and measure the final state:** We iteratively apply the oracle and the diffuser circuit to take the present superposition closer and closer to the goal state. Measurement of this state along the computational basis gives us a bitstring, which with some constant probability is the solution x such that $f(x) = 1$.

Ampliture Amplification algorithms : The algorithmic structure proposed by Grover can be extended to solve other problems, the notion being that a quantum circuit that finds an algorithm with probability p needs to be run only $O(1/\sqrt{p})$ times and not $O(1/p)$ times [7].

2.2.1.2 Shor's Algorithm

Shor's Algorithm is used to perform factorization of an integer product of two large prime numbers [44].

The problem : Given an integer that is known to be the product of two large prime numbers, we have to factor the number back into the composing integers.

Overview of the Algorithm : Following is a brief explanation of how the Shor's Algorithm works:

1. **Preparing the Unitary**: We need to model a unitary operator which performs modular multiplication by some constant a.

$$U \left| y \right\rangle = \left| ay \mod N \right\rangle \tag{2.7}$$

The n-qubit states for our quantum system are being represented as $\left| 0 \right\rangle, \left| 1 \right\rangle, \cdots \left| N - 1 \right\rangle$, on the qubits they are being represented as the bitstrings corresponding to those numbers.

2. **Period Finding to Phase Estimation**: Generate a linear combination of states which is an eigenstate of the unitary operator. The states in the linear combination will form an r-element subgroup, and the phase terms multiplied to each state will be the r-th roots of unity.

$$\left| u_s \right\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-\frac{2\pi i s k}{r}} \left| a^k \mod N \right\rangle \tag{2.8}$$

$$U \left| u_s \right\rangle = e^{\frac{2\pi i s}{r}} \left| u_s \right\rangle \tag{2.9}$$

3. **Period Finding to Factorization**: We compute the phase $\frac{s}{r}$ from the process above. Using this, we can generate a fractional approximation to the phase and estimate the value of r. Now we know that

$$a^r \mod N = 1 \tag{2.10}$$

With high probability, r will be even, if it's not we will repeat this process with a new a. Given the above, we can guess a factor of N as $a^{r/2} - 1$ or $a^{r/2} + 1$.

$$(a^{r/2} - 1)(a^{r/2} + 1) \mod N = 0 \tag{2.11}$$

Having ensured initially that a is not a factor of N, we have found a high-probability guess for the factors being one of these two numbers. We repeat this process until this guess turns out to be correct.

Other Comments : Shor's Algorithm provides an exponential advantage over any known classical algorithm, but at the moment, it demands too many qubits, and it demands that fault tolerance be implemented, which is presently infeasible. An entire class of Quantum Algorithms takes inspiration from this use of period finding using the Quantum Phase Estimation process, and one prominent example is the HHL algorithm for solving linear systems.

2.2.1.3 Hamiltonian Simulation

Nature is hard to simulate on a classical computer. Writing down quantum wavefunctions and updating them is computationally very expensive. As Richard Feynman said: *"Nature is not classical, dammit, and if you want to make a simulation of nature, you'd better make it quantum mechanical, and by golly, it's a wonderful problem, because it doesn't look so easy."* [14].

The most natural solution is to use the quantum systems to emulate other quantum systems and perform these computations. It seems intuitive that quantum computers are efficient at doing quantum simulations; all that needs to be done is that the state of an arbitrary system needs to be mapped onto that of ours.

There are many ways of performing Hamiltonian simulation on general-purpose gate-based quantum computers. Some of those methods are listed below:

- Using Trotterization: Given a hamiltonian which can be represented as a linear combination of local Hamiltonians, $H = A + B + C$, the trotter decomposition of Unitary corresponding to the Hamiltonian can be approximated by

$$U = e^{-iHt} = (e^{-iAt/r}e^{-iAt/r}e^{-iAt/r})^r \qquad (2.12)$$

For larger values of r this results in a pretty accurate simulation.

- Using Taylor Expansion of Hamiltonian: The Hamiltonian can be expanded out as a sum of unitaries, and then the corresponding unitary can be taylor-series expanded out as a linear combination of the unitaries corresponding to terms of the Taylor expansion.

$$U = e^{-iHt} = \sum_{n=0}^{\infty} \frac{(-iHt)^n}{n!} \qquad (2.13)$$

- Using Quantum Walks: Quantum walks can be constructed corresponding to the Hamiltonian such that the spectrum of the graph on which the walk is constructed is such that its eigenvalues correspond to the ground state of the target Hamiltonian.

- Using Quantum Signal Processing: The QSP algorithm transduces eigenvalues of the Hamiltonian into an ancilla, then transforms the eigenvalues with single-qubit rotations, and finally projects it back to the ancilla.

2.2.2 Variational Circuits

In the previous section, we have discussed purely quantum algorithms, which possess an advantage over their presently classical competitors but need fault-tolerant quantum computers with many qubits to be executed.

In this section, we look at a few hybrid quantum-classical optimization algorithms, called variational quantum algorithms.

Each optimization problem is solved over a parameter space and attempts to maximize some metric which we call the loss. For a quantum circuit, we can visualize the loss function as some hamiltonian for which we are trying to find the lowest energy state, i.e., the ground eigenstate. Some parametrized circuit generates this ground eigenstate, and a classical optimizer is responsible for finding as low an energy state as possible.

2.2.2.1 Quantum Approximate Optimization Algorithm
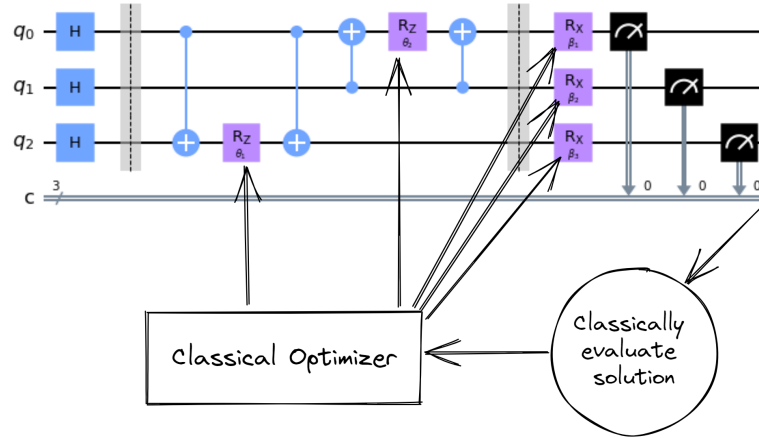
Figure 2.4: An example of a QAOA circuit with $p = 1$ blocks and generated to compute max-cut on a graph with 3-nodes and 2-edges between (0, 2) and (1, 0). The state generated by the circuit is parameterized in terms of the five parameters in the circuit. The output along the z-axis is measured and used to compute the classical loss value, i.e., the size of the max-cut given to which each node (represented by a qubit) belongs. Finally, a classical optimizer is used to reduce the value of the loss function such that

Quantum Approximate Optimization Algorithms are a class of variational quantum algorithms used as a general-purpose hammer to solve combinatorial optimization problems probabilistically.

Motivation for the Algorithm : The Algorithm takes inspiration from adiabatic quantum computing, which is a method that does the following: We start with a quantum state that is an easily preparable ground state of some Hamiltonian, and we vary this Hamiltonian sufficiently slowly and try to reach

some target Hamiltonian, then the given quantum state will also track the ground state of the current form of the Hamiltonian. So the result of the Algorithm, if performed slowly enough, will be the ground state of the target Hamiltonian the preparation of which might not have been obvious otherwise.

QAOA takes that philosophy to the gate-computation model and uses the trotter decomposition to approximate the application of unitaries corresponding to these Hamiltonians for sufficiently short periods of time.

Algorithm   : The following steps are performed to execute QAOA for a given optimization problem.

1. **Prepare the initial state**: We start by applying a Hadamard on all qubits to create an initial state which is an equal superposition of all basis states.

2. **Apply the mixing unitary**: The mixing unitary corresponds to the mixing Hamiltonian, which is the Hamiltonian for which the initial state we have prepared is a ground state.

$$U(H_B) = e^{-i\beta H_B} \tag{2.14}$$

    The mixer hamiltonian, therefore, is $X^{\otimes N}$, and the corresponding unitary becomes $R_x(\beta)^{\otimes N}$.

3. **Apply the problem Hamiltonian**: The problem unitary corresponds to the Hamiltonian formulation of our problem, and this is the Hamiltonian for which we are attempting to find the ground eigenstate.

$$U(H_P) = e^{-i\gamma H_P} \tag{2.15}$$

    This has to be phrased for each problem.

4. **Repeat for p iterations**: We present the mixing-Hamiltonian problem-Hamiltonian process for p-iterations. Each player is a β-duration mixing Hamiltonian and a γ-duration problem Hamiltonian application. For tuned values of β and γ and sufficiently large p, this can be made to approximate adiabatic computing.

5. **Measure and classically compute loss**: The loss value that we are attempting to optimize is the following:

$$\mathcal{L} = \langle \psi(\beta, \gamma) | H_p | \psi(\beta, \gamma) \rangle \tag{2.16}$$

    To compute this, we measure all qubits, and using the bitstring obtained in our classical registers, we classically compute the value of the optimization metric for this bitstring solution.

6. **Classically compute derivatives and update parameters**: The parameters for each of the p layers (β, γ) are optimized using some method like gradient descent or adam. We attempt to reduce the value of the loss, as defined by the parameter updates in equation 2.16.

QAOA for Max Cut of a Graph : Given a graph with vertices $v \in V$ and edges $e \in E \subset V \times V$, we want to compute a partition of vertices into two sets $f : V \to \{-1, +1\}$ such that the number of edges going across the partition ($C = \{e = (u, v) | e \in E \; ; \; f(u) \neq f(v)\}$) is maximized ($\max_f |C|$).

The QAOA formulation follows naturally. The classical loss function is the following:

$$\mathcal{L} = \frac{-1}{2} \sum_{e=(u,v) \in E} (f(u) \cdot f(v) - 1) \tag{2.17}$$

Since the Pauli-Z operator on the $|0\rangle$ and $|1\rangle$ state acts similar to the function f, in the quantum phrasing we can equivalently use the Z operators. We can drop the global constants and additive terms when writing out the optimization objective.

$$\mathcal{H} = \frac{-1}{2} \sum_{e=(u,v)} I_0 \otimes I_1 \otimes \ldots \otimes Z_u \otimes \ldots \otimes Z_u \otimes \ldots \otimes I_N \tag{2.18}$$

This can be efficiently implemented in a circuit, which will be used as the problem unitary.

Other applications : The QAOA algorithm can be further used to solve approximately other problems, including those that are NP-complete such as Weighted Max-Cut, 3-SAT, Travelling Salesman problem, amongst others. On the problem of Max-Cut, QAOA had, for a short span of about 3-months, held the best approximation ratio until it was superseded by semi-definite programming. Even though QAOA may not be able to be within the bounds achievable by classical computers, the universality of its applicability is remarkable. Coming up with classical methods to outperform QAOA on every new task takes much effort, semi-definite programming solutions have proven to be the best bet to outperform QAOA, but they have to be formulated afresh for each problem. Therefore, as a general-purpose solution to many problems, QAOA stands as a very good bet.

2.2.3 Variational Quantum Eigensolvers

Variational Quantum Algorithms is another entry in the class of variation algorithms, which tries to estimate the ground eigenstate and eigenvalues of a Hamiltonian using the method of optimization described above.

$$\lambda_{min} \leq \langle H \rangle_\psi = \langle \psi | H | \psi \rangle = \sum_{i=1}^{N} \lambda_i | \langle \psi_i | \psi \rangle |^2 \tag{2.19}$$

2.3 Reinforcement Learning

2.3.1 What is Reinforcement Learning

Machine Learning and all associated sub-disciplines are motivated by the goal of achieving artificial general intelligence, that is, being able to mimic the human mind and even surpass its capacity to

perceive, compute and actuate. The human mind deals with various problems differing greatly in their phrasing, the solutions they admit, etc. Of this host of problem types,

Deep Learning is an extremely powerful and popular one of these methods, which uses parameterized function approximators (aka. neural networks) to learn arbitrary functions directly from examples. We typically learn functions that take as input numerical data and associated structure (e.g., graphs) and produce one or many continuous-valued outputs (regression) or discrete-value outputs (classification). This has been employed with great success in computational chemistry, for instance, in predicting properties of molecules like solubility, smell, energy, etc.

Despite all their predictive power, these methods are limited in the problems they can solve. One limitation is our inability to provide many labeled examples since running laboratory experiments or expensive in-silico simulations are often too time and resource-consuming. Another issue is that the output may not be a simple function of its inputs. For instance, when predicting molecular coordinates from molecular graphs, our outputs depend greatly on each other the position of one atom affects that of all others, and therefore a single step function cannot solve such a problem; an iterative approach to optimize these coordinates is required. In such cases where a problem is solved in many steps, there is no notion of the correct result after a single step; we can only score if the composite of steps produces the final result. All these problems necessitate a machine learning method that can produce outputs over several timesteps and be able to reason about the correctness of its outputs based on rewards it may obtain at a different time in our process. This method is Reinforcement Learning. [33]

2.3.1.1 Markov Decision Processes

A Markov Decision process is any real or simulated process going on in time where each decision follows the Markovian Property, i.e., any future state transitions or rewards are conditionally independent of the past states and actions given the present state the environment is in.

A Markov Decision Process (MDP) can be represented as a tuple $\langle S, A, T_a(s, s'), R_a(s, s') \rangle$, where S is the set of all states, A is the set of all actions available from any given state, $T_a(s, s')$ is the transition model which represents the probability of going from a starting state s to a next state s' given that the action a was taken, and $R_a(ss')$ is the reward obtained when this transition is realized.

Reinforcement Learning is a method of solving Markov Decision Processes. For our problem to be solved by RL, we need to ensure that our formulation is Markovian, i.e., our state has enough information to, given the action, predict the probability of the next state and the associated reward.

2.3.1.2 Value Function and Policy Function

At every point in time, our agent has access to the state and gets to choose an action. For this action, it receives a reward, and the state of the simulation is updated. This process continues indefinitely until a terminal state is reached, i.e., one where no further progress needs to be made and no future rewards can be collected. This entire trajectory of states and actions together comprises an episode.

17

The agent maintains a function which is called its **policy function** $\pi(s, a)$, which given the current state, gives the probability of each action it can take from that state. Our agent is allowed to be stochastic for various practical and theoretical reasons, so the probability for more than one action in a given state is allowed to be non-zero. This is the function that we shall attempt to optimize while learning from our environment.

While acting according to any policy function, we can associate each state with what we call the **value function** $V_\pi(s)$, which represents the expected sum of rewards till the end of the episode obtainable by following the policy. The optimal policy function π leads to the maximum value function for the starting state.

Value-function of one state can be written in terms of that of others. To compute these values over all the states, we need to apply our updates iteratively.

$$V(s) = \sum_{a \in A} \pi(s, a) \sum_{s'} T_a(s, s')(V(s') + R_a(s, s')) \tag{2.20}$$

Instead of associating a value with each state, we can associate it with a state-action pair. This function is called the Q-function, and it carries equivalent information to the value function.

$$Q(s, a) = \sum_{s'} T_a(s, s') \left(R(s, s') + V(s') \right) \tag{2.21}$$

$$= \sum_{s'} T_a(s, s') \left(R(s, s') + \sum_{a \in A} \pi(s', a) Q(s', a) \right) \tag{2.22}$$

2.3.2 Reinforcement Learning Algorithms

In the following sections, we shall see three kinds of models:

- Value Function Optimizers

- Policy Function Optimizers

- Actor-Critic Systems

- Planning based Reinforcement Learning

2.3.2.1 Deep Q-Networks

The first class of models attempts to approximate the value function. Assuming that our policy function will be that which is optimal, and assuming that our actions are deterministic (i.e., transition probabilities are 1 for the state we result in after an action and 0 otherwise), we can rewrite equation 2.23 as:

$$Q(s, a) \leftarrow R(s, s') + \max_{a \in A} Q(s', a) \tag{2.23}$$

For almost all problems in the real world, the state space is too large to maintain explicitly. Therefore we use a parameterized function Q_θ, typically a neural network, to approximate the q-value from any given state-action pair.

The parameters θ can be updating using gradient based methods. The update operation in equation 2.24 is

$$\theta_{k+1} = \theta_k - \alpha \nabla_\theta \left[\frac{1}{2} \Big(Q_\theta(s,a) - \Big(R(s,a,s') + \gamma \max_{a'} Q_{\theta_k}(s',a') \Big) \Big) \right] \Bigg|_{\theta_k} \qquad (2.24)$$

Several improvements to the training efficiency and stability of the DQN algorithm have been made; a few examples are the Double DQN by [53]. This set of improvements put together has been analyzed by [19] under the name Rainbow DQN.

2.3.2.2 Policy Function Approximators

The policy function $\pi_\theta(s,a)$ gives the probability of each action given the state. In value function methods, we computed the policy by finding the action with the maximum expected value and assigning it a probability of 1 and other actions 0 for each state. When learning the policy directly, we use a stochastic policy instead, which chooses smooth and optimizable actions.

2.3.2.2.1 Reasons to use policy gradients:

1. Learning value function may be much harder than learning the relative quality of actions. E.g., given the task of designing molecules with high solubility and a procedure that keeps adding bonds iteratively, it can be very hard to predict the expected solubility of the molecule formed at the end of trajectories (value function) while predicting that adding a highly polar bond is more beneficial than adding non-polar ones (policy function).

2. We might want to obtain an inherently stochastic policy, where policy-based methods are the better choice. One example is when designing molecules with certain properties, and we want a stochastic policy to sample different molecules that optimize on the target property and then rank them based on synthetic ease or the such.

3. Many a time, the action space is continuous or intractably large, and maximizing the value over all the actions is not feasible. Here we can only use policy-based methods. Geometry optimization is one example where the action is predicting the molecular coordinates of a single atom, which leads to an un-countably infinite-sized action space.

2.3.2.2.2 Method : To optimize our policy, we sample trajectories from our policy and increase the probability of actions in trajectories that obtain high rewards and lower the probability of those with lesser rewards.

The utility of our policy is the expected reward under trajectories sampled from this policy; this is the quantity we wish to maximize over the parameters θ. To perform this maximization, we compute $\nabla_\theta U(\theta)$ and update the parameter vector as $\theta \leftarrow \theta + \epsilon \nabla_\theta U(\theta)$. The gradient only depends on the gradient of the log of our policy function scaled by the rewards obtained along the trajectory and, very importantly, does not depend on the true transition model. Equation 2.25 follows from a mathematically involved derivation done in [].

$$\nabla_\theta U(\theta) \leftarrow \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)}) \left(\sum_{k=t}^{H-1} R(s_k^{(i)}, u_k^{(i)}) - b(s_t^{(i)}) \right) \qquad (2.25)$$

2.3.2.2.3   Other Nuances   : Despite having the gradient that we need to update along, it is unclear what learning rate we should use to perform the said update. Unlike in deep learning, where the next iteration would correct if we overstep along the gradient, an overstep in our policy can lead to evaluation over an incorrect policy and can essentially wipe out all we have learned till now. Trust Region policy optimizations (TRPO) by [41] and Proximal Policy Optimizations (PPO) by [43] are methods that address this. Furthermore, to increase sample efficiency, Direct Deterministic Policy Gradients (DDPG) by [25], and Soft Actor critic (SAC) [17] are used. These methods have not seen great application in chemistry but hold great promise given their popularity in other reinforcement learning sub-domains.

2.3.2.3   Actor Critic Methods

In equation 2.25, we are free to subtract a baseline value $b(s_t^{(i)})$ from the summed up rewards for each action; however, this baseline should be independent of the action and can only depend on the state. Subtraction of this baseline leads to lower variance estimates in the value of actions. The network now has to predict a quantity called the advantage. Advantage represents the difference in the value of each action over or below the expected reward obtainable from the state.

$$A(s, a) = Q(s, a) - V(s) \qquad (2.26)$$

This is implemented in practice using two networks, an actor network, which estimates the values of the actions, and a critic network, which estimates the resultant values of the states, which we subtract as a baseline from the rewards. These methods are often known to be stabler than their pure policy-gradient counterparts.

There are several variants on how the critic network and the explicit rollout together lead to the estimate of the value for each state, which has been discussed in detail by [30, 31, 42]

2.3.2.4   Monte Carlo Tree Search

When the transition model (next state and reward given action) is known, we can plan explicitly using a tree search. Since the tree would grow combinatorially big (molecule generation via such means

would have every possible molecule and its substructure as a node in the tree), we use reinforcement learning to find the most promising nodes. Monte Carlo Tree Search is one such method, which has gained prominence due to its use in AlphaGo by [45] to play Go and in AlphaZero by [47] to play Chess, Go, and other games with no human supervision during training.

MCTS has been used extensively in chemistry wherever the exact transition model is known, in problems like Molecule Generation and Reaction path prediction.

Chapter 3

qRoute: Qubit Routing using Graph Neural Network aided Monte Carlo Tree Search

3.1 Abstract

Near-term quantum hardware can support two-qubit operations only on the qubits that can interact with each other. Therefore, to execute an arbitrary quantum circuit on the hardware, compilers have to first perform the task of qubit routing, i.e., to transform the quantum circuit either by inserting additional SWAP gates or by reversing existing CNOT gates to satisfy the connectivity constraints of the target topology. The depth of the transformed quantum circuits is minimized by utilizing the Monte Carlo tree search (MCTS) to perform qubit routing by making it both construct each action and search over the space of all actions. It is aided in performing these tasks by a Graph neural network that evaluates the value function and action probabilities for each state. Along with this, we propose a new method of adding mutex-lock like variables in our state representation which helps factor in the parallelization of the scheduled operations, thereby pruning the depth of the output circuit. Overall, our procedure (referred to as QRoute) performs qubit routing in a hardware agnostic manner, and it outperforms other available qubit routing implementations on various circuit benchmarks.

3.2 Introduction

The present-day quantum computers, more generally known as Noisy Intermediate-Scale quantum (NISQ) devices [39]

To execute an arbitrarily given quantum circuit on the target quantum hardware, a compiler routine must transform it to satisfy the connectivity constraints of the topology of the hardware [11]. These transformations usually include the addition of SWAP gates and the reversal of existing CNOT gates. This ensures that any non-local quantum operations are performed only between the qubits that are nearest-neighbors. This process of circuit transformation by a compiler routine for the target hardware is known as qubit routing [11]. The output instructions in the transformed quantum circuit should follow

(a) Quantum circuit   (b) Decomposed circuit   (c) Decomposed transformed circuit
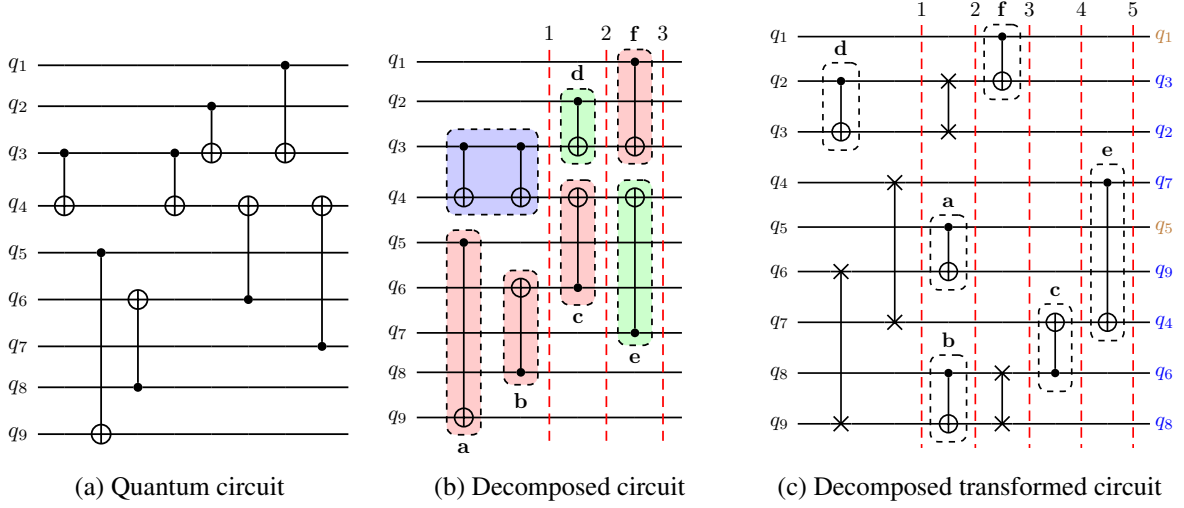
Figure 3.1: An example of qubit routing on a quantum circuit for 3×3 grid architecture (Figure 3.2a). (a) For simplicity, the original quantum circuit consists only of two-qubit gate operations. (b) Decomposition of the original quantum circuit into series of slices such that all the instructions present in a slice can be executed in parallel. The two-qubit gate operations: $\{d, e\}$ (green) comply with the topology of the grid architecture whereas the operations: $\{a, b, c, f\}$ (red) do not comply with the topology (and therefore cannot be performed). Note that the successive two-qubit gate operations on $q_3 \rightarrow q_4$ (blue) are redundant and are not considered while routing. (c) Decomposition of the transformed quantum circuit we get after qubit routing. Four additional SWAP gates are added that increased the circuit depth to 5, i.e., an overhead circuit depth of 2. The final qubit labels are represented at the end right side of the circuit. The qubits that are not moved (or swapped) are shown in brown ($\{q_1, q_5\}$), while the rest of them are shown in blue.

the connectivity constraints and essentially result in the same overall unitary evolution as the original circuit [38].

In the context of NISQ hardware, this procedure is of extreme importance as the transformed circuit will, in general, have higher depth due to the insertion of extra SWAP gates. This overhead in the circuit depth becomes more prominent due to the high decoherence rates of the qubits and it becomes essential to find the most optimal and efficient strategy to minimize it [11, 18, 38]. In this article, we present a procedure that we refer to as *QRoute*. We use Monte Carlo tree search (MCTS), which is a look-ahead search algorithm for finding optimal decisions in the decision space guided by a heuristic evaluation function [21, 34, 22]. We use it for explicitly searching the decision space for depth minimization and as a stable and performant machine learning setting. It is aided by a Graph neural network (GNN) [58], with an architecture that is used to learn and evaluate the heuristic function that will help guide the MCTS.
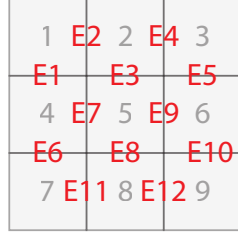
(a) 3×3 grid architecture with edges (i.e. neighboring qubits) labelled
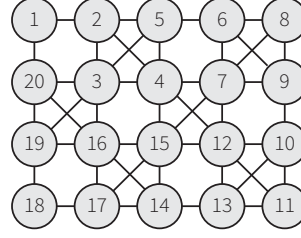
(b) IBMQX-20 architecture represented as a graph

Figure 3.2: Examples of qubit connectivity graphs for some common quantum architectures

3.3 Qubit Routing

In this section, we begin by defining the problem of qubit routing formally and discussing the work done previously in the field.

3.3.1 Describing the Problem

The topology of quantum hardware can be visualized as a qubit connectivity graph (Fig. 3.2). Each node in this graph would correspond to a physical qubit which in turn might correspond to a logical qubit. The quantum instruction set, which is also referred to as quantum circuit (Fig. 3.1a), is a sequential series of single-qubit and two-qubit gate operations that act on the logical qubits. The two-qubit gates such as CNOT can only be performed between two logical qubits iff there exists an edge between the nodes that correspond to the physical qubits, [18]. This edge could be either unidirectional or bidirectional, i.e., CNOT can be performed either in one direction or in both directions. In this work, we consider only the bidirectional case, while noting that the direction of a CNOT gate can be reversed by sandwiching it between a pair of Hadamard gates acting on both control and target qubits [15].

Given a target hardware topology $\mathcal{D}$ and a quantum circuit $\mathcal{C}$, the task of qubit routing refers to transforming this quantum circuit by adding a series of SWAP gates such that all its gate operations then satisfy the connectivity constraints of the target topology (Fig. 3.1c). Formally, for a routing algorithm R, we can represent this process as follows:

$$R(\mathcal{C},\ \mathcal{D}) \rightarrow \mathcal{C}' \tag{3.1}$$

Depth of $\mathcal{C}'$ (transformed quantum circuit) will, in general, be more than that of the original circuit due to the insertion of additional SWAP gates. This comes from the definition of the term *depth* in the context of quantum circuits. This can be understood by decomposing a quantum circuit into series of individual slices, each of which contains a group of gate operations that have no overlapping qubits, i.e., all the instructions present in a slice can be executed in parallel (Fig. 3.1b). The depth of the quantum circuit then refers to the minimum number of such slices the circuit can be decomposed into, i.e., the minimum

amount of parallel executions needed to execute the circuit. The goal is to minimize the overhead depth of the transformed circuit with respect to the original circuit.

This goal involves solving two subsequent problems of (i) qubit allocation, which refers to the mapping of program qubits to logic qubits, and (ii) qubit movement, which refers to routing qubits between different locations such that interaction can be made possible [52]. In this work, we focus on the latter problem of qubit movement only and refer to it as qubit routing. However, it should be noted that qubit allocation is also an important problem and it can play an important role in minimizing the effort needed to perform qubit movement.

3.3.2 Related Work

The first major attraction for solving the qubit routing problem was the competition organized by IBM in 2018 to find the best routing algorithm. This competition was won by [66], for developing a Computer Aided Design-based (CAD) routing strategy. Since then, this problem has been presented in many different ways. These include graph-based architecture-agnostic solution by [11], showing equivalence to the travelling salesman problem by [37], machine learning based methods by [36], and heuristic approaches by [55], [6], [9], etc. A reinforcement learning in a combinatorial action space solution was proposed by [18], which suggested used simulated annealing to search through the combinatorial action space, aided by a Feed-Forward neural network to judge the long-term expected depth. This was further extended to use Double Deep Q-learning and prioritized experience replay by [38].

Recently, Monte Carlo tree search (MCTS), a popular reinforcement learning algorithm [8] previously proven successful in a variety of domains like playing puzzle games such as Chess and Go [46], and was used by [63] to develop a qubit routing solution.

3.3.3 Our Contributions

Our work demonstrates the use of MCTS on the task of Qubit Routing and presents state of the art results. Following are the novelties of this approach:

- We use an array of mutex locks to represent the current state of parallelization, helping to reduce the depth of the circuits instead of the total quantum volume, in contrast to previous use of MCTS for qubit routing in [63].

- The actions in each timestep (layer of the output circuit) belong to a innumerably large action space. We phrase the construction of such actions as a Markov decision process, making the training stabler and the results better, particularly at larger circuit sizes, than those obtained by performing simulated annealing to search in such action spaces [18, 38]. Such approach should be applicable to other problems of a similar nature.

- Graph neural networks are used as an improved architecture to help guide the tree search.

Finally, we provide a simple python package containing the implementation of QRoute, together with an easy interface for trying out different neural net architectures, combining algorithms, reward structures, etc.
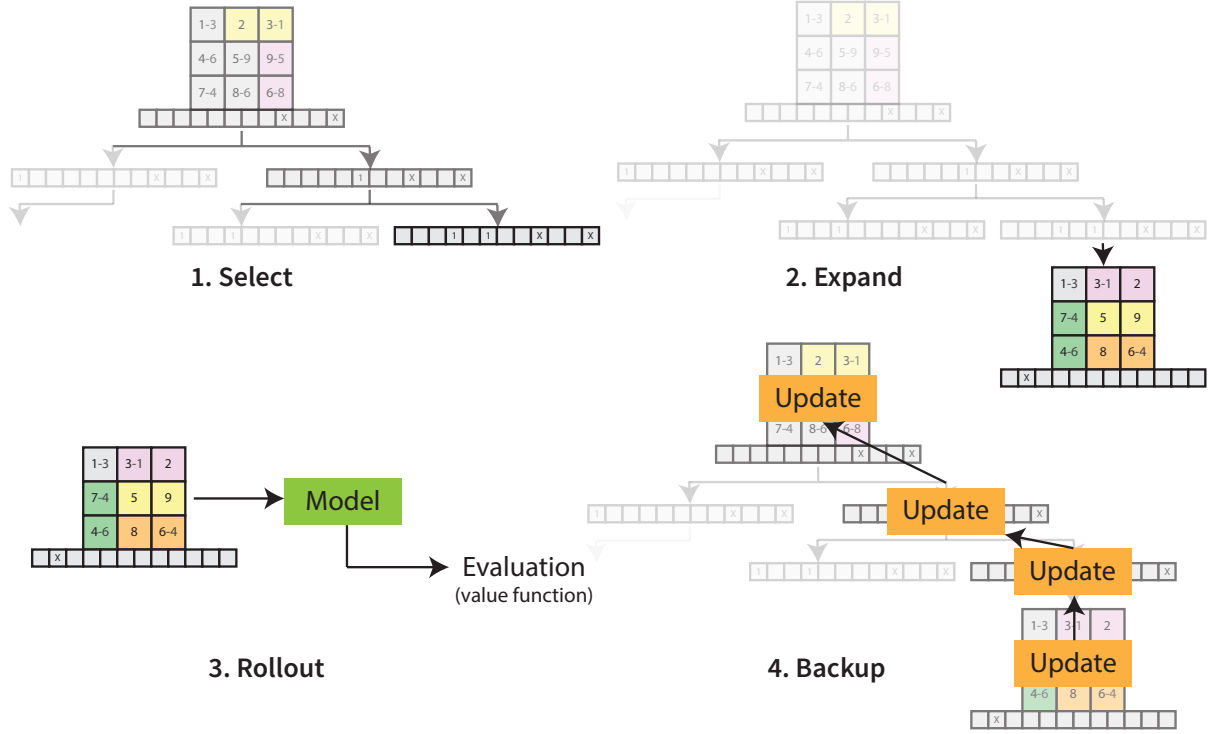
Figure 3.3: Iteration of a Monte Carlo tree search: (i) select - recursively choosing a node in the search tree for exploration using a selection criteria, (ii) expand - expanding the tree using the available actions if an unexplored leaf node is selected, (iii) rollout - estimating long term reward using a neural network for the action-state pair for the new node added to search tree, and (iv) backup - propagating reward value from the leaf nodes upwards to update the evaluations of its ancestors.

3.4   Method

The QRoute algorithm takes in an input circuit and an injective map, $\mathcal{M} : Q \rightarrow N$, from logical qubits to nodes (physical qubits). Iteratively, over multiple timesteps, it tries to schedule the gate operations that are present in the input circuit onto the target hardware. To do so, from the set of unscheduled gate operations, $\mathcal{P}$, it takes all the current operations, which are the first unscheduled operation for both the qubits that they act on, and tries to make them into local operations, which are those two-qubit operations that involve qubits that are mapped to nodes connected on the target hardware.

In every timestep t, QRoute starts by greedily scheduling all the operations that are both current and local in $\mathcal{P}$. To evolve $\mathcal{M}$, it then performs a Monte Carlo tree search (MCTS) to find an optimal set of SWAPs by the evaluation metrics described in the Section 3.4.2 such that all operations in the current

timestep put together form a parallelizable set, i.e., a set of local operations such that no two operations in the set act on the same qubit. The number of states we can encounter in the action space explodes exponentially with the depth of our search, therefore an explicit search till the circuit is done compiling is not possible. Therefore we cut short our search at some shallow intermediate state, and use a neural network to get its heuristic evaluation.

The following subsections describe in greater detail the working of the search and the heuristic evaluation.

3.4.1 State and Action Space

Definition 3.4.1 (State) *It captures entire specification of the state of compilation at some timestep t. Abstractly, it is described as:*

$$s_t = (\mathcal{D}, \ \mathcal{M}_t, \ \mathcal{P}_t, \ \mathcal{L}_t) \tag{3.2}$$

where, $\mathcal{D}$ is the topology of target hardware, and $\mathcal{M}_t$ and $\mathcal{P}_t$ represents the current values of $\mathcal{M}$ and $\mathcal{P}$ respectively. $\mathcal{L}_t$ is the set of nodes that are locked by the gate operations from the previous timestep and therefore cannot be operated in the current timestep.

Definition 3.4.2 (Action) *It is a set of SWAP gates (represented by the pair of qubits it acts on) such that all gates are local, and its union with the set of operations that were scheduled in the same timestep forms a parallelizable set.*

We are performing a tree search over state-action pairs. Since the number of actions that can be taken at any timestep is exponential in the number of connections on the hardware, we are forced to build a single action up, step-by-step.

Definition 3.4.3 (Move) *It is a single step in a search procedure which either builds up the action or applies it to the current state. Moves are of the following two types:*

1. *SWAP(n_1, n_2): Inserts a new SWAP on nodes n_1 and n_2 into the action set. Such an insertion is only possible if the operation is local and resulting set of operations for the timestep form a parallel set.*

2. *COMMIT: Finishes the construction of the action set for that timestep. It also uses the action formed until now to update the state s_t (schedules the SWAP gates on the hardware), and resets the action set for the next step.*

In reality, different gate operations take different counts of timesteps for execution. For example, if a hardware requires SWAP gate to be broken down into CNOT gates, then it would take three timesteps for complete execution [15]. This means, operations which are being scheduled must maintain mutual exclusivity with other other operations over the nodes which participates in them. This is essential to minimizing the depth of the circuit since it models paralleizability of operations.

However, constructing a parallelizable set and representing the state of parallelization to our heuristic evaluator is a challenge. But an analogy can be drawn here to the nodes being thought of as "resources" that cannot be shared, and the operations as "consumers" [13]. This motivates us to propose the use of Mutex Locks for this purpose. These will lock a node until a scheduled gate operation involving that node executes completely. Therefore, this allows our framework to naturally handle different types of operations which take different amounts of time to complete.

For every state-action pair, the application of a feasible move m on it will result in a new state-action pair: $(s, a) \xrightarrow{m} (s', a')$. This is a formulation of the problem of search as a Markov Decision Process. Associated with each such state-action-move tuple $((s, a), m)$, we maintain two additional values that are used by MCTS:

1. *N-value* - The number of times we have taken the said move m from said state-action pair (s, a).

2. *Q-value* - Given a reward function $\mathcal{R}$, it is the average long-term reward expected after taking said move m over all iterations of the search. (Future rewards are discounted by a factor γ)

$$Q((s, a), m) = \mathcal{R}((s, a), m) + \gamma \frac{\sum_{m'} N((s', a'), m') \cdot Q((s', a'), m')}{\sum_{m'} N((s', a'), m')} \tag{3.3}$$

3.4.2 Monte Carlo Tree Search

Monte Carlo tree search progresses iteratively by executing its four phases: select, expand, rollout, and backup as illustrated in Fig. 3.3. In each iteration, it begins traversing down the existing search tree by selecting the node with the maximum UCT value (Eq. 3.4) at each level. During this traversal, whenever it encounters a leaf node, it expands the tree by choosing a move m from that leaf node. Then, it estimates the scalar evaluation for the new state-action pair and backpropagates it up the tree to update evaluations of its ancestors.

To build an optimal action set, we would want to select the move m with the maximum true Q-value. But since true Q-values are intractably expensive to compute, we can only approximate the Q-values through efficient exploration. We use the Upper Confidence Bound on Trees (UCT) objective [22] to balance exploration and exploitation as we traverse through the search tree. Moreover, as this problem
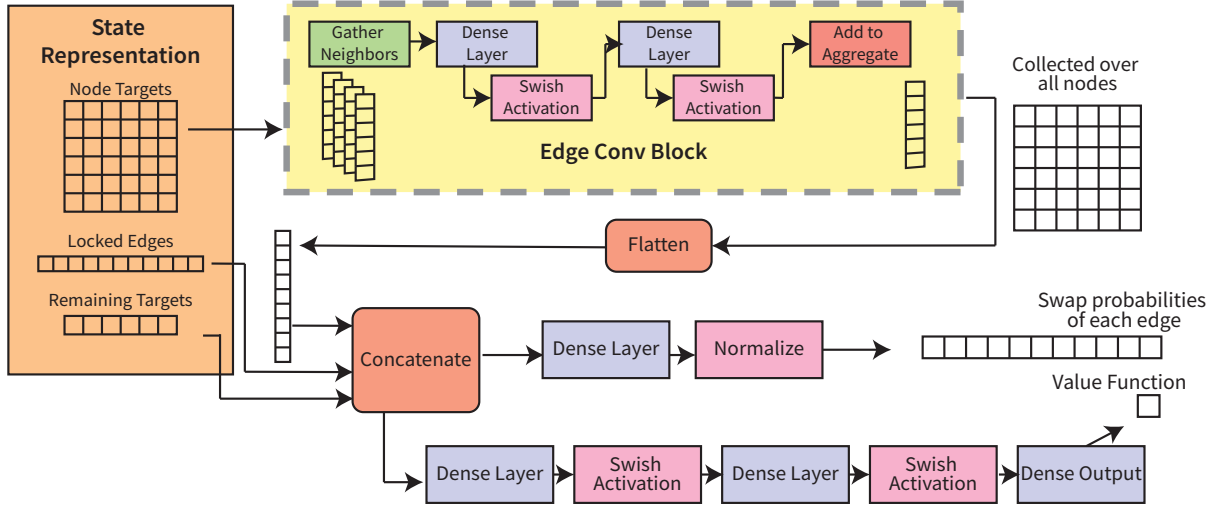
Figure 3.4: Graph neural network architecture that approximates the value function and the policy function.

results in a highly asymmetric tree, since some move block a lot of other moves, while others block fewer moves, we use the formulation of UCT adapted for asymmetric trees [32]:

$$
\text{UCT}((s,a),m) = Q((s,a),m) + \\
c\frac{\sqrt{\sum_m N((s,a),m)}}{N((s,a),m)} \times p(m|(s,a))
$$
(3.4)

Here, the value $p(m|(s,a))$ is the prior policy function, which is obtained by adding a Dirichlet noise to the policy output of the neural network [48]. As MCTS continues probing the action space, it gets a better estimate of the true values of the actions. This means that it acts as a policy enhancement function whose output policy (Eq. 3.5) can be used to train the neural network's prior (π), and the average Q-value computed can be used to train its scalar evaluation (Eq. 3.6).

$$
\pi(m|(s,a)) \propto N((s,a),m))
$$
(3.5)

$$
\mathcal{V}((s,a)) = \frac{\sum_m Q((s,a),m)}{\sum_m N((s,a),m)}
$$
(3.6)

The details of how MCTS progresses have been elaborated in the supplementary. Once it gets terminated, i.e., the search gets completed, we go down the tree selecting the child with the maximum Q-value at each step until a COMMIT action is found, we use the action set of the selected state-action pair to schedule SWAPs for the current timestep, and we re-root the tree at the child node of the COMMIT action to prepare for the next timestep.

3.4.3 Neural Network Architecture

Each iteration of the MCTS requires evaluation of Q-values for a newly encountered state-action pair. But these values are impossible to be computed exactly since it would involve an intractable number of iterations in exploring and expanding the complete search tree. Therefore, it is favorable to heuristically evaluate the expected long-term reward from the state-action pair using a Neural Network, as it acts as an excellent function approximator that can learn the symmetries and general rules inherent to the system.

So, once the MCTS sends a state-action pair to the evaluator, it begins by committing the action to the state and getting the resultant state. We then generate the following featurized representation of this state and pass this representation through the neural-network architecture as shown in Fig. 3.4.

1. *Node Targets* - It is a square boolean matrix whose rows and columns correspond to the nodes on a target device. An element (i, j) is true iff some logical qubits q_x and q_y are currently mapped to nodes i and j respectively, such that (q_x, q_y) is the first unscheduled operation that q_x partakes in.

2. *Locked Edges* - It is a set of edges (pairs of connected nodes) that are still locked due to either of its qubits being involved in an operation in the current timestep or another longer operation that hasn't yet terminated from the previous timesteps.

3. *Remaining targets* - It is a list of the number of gate-operations that are yet to be scheduled for each logical qubit.

The SWAP operations each qubit would partake in depends primarily on its target node, and on those of the nodes in its neighborhood that might be competing for the same resources. It seems reasonable that we can use a Graph Neural Network with the device topology graph for its connectivity since the decision of the optimal SWAP action for some node is largely affected by other nodes in its physical neighborhood. Therefore, our architecture includes an edge-convolution block [58], followed by some fully-connected layers with Swish [40] activations for the policy and value heads. The value function and the policy function computed from this neural network are returned back to the MCTS.

3.5 Results

We compare QRoute against the routing algorithms from other state-of-the-art frameworks on various circuit benchmarks: (i) Qiskit and its three variants [2]: (a) basic, (b) stochastic, and (c) sabre, (ii) Deep-Q-Networks (DQN) from [38], (iii) Cirq [10], and (iv) t|ket⟩ from Cambridge Quantum Computing (CQC) [50]. Qiskit's transpiler uses gate commutation rules while perform qubit routing. This strategy is shown to be advantageous in achieving lower circuit depths [20] but was disabled in our simulations to have a fair comparison. The results for DQN shown are adapted from the data provided by the authors [38].
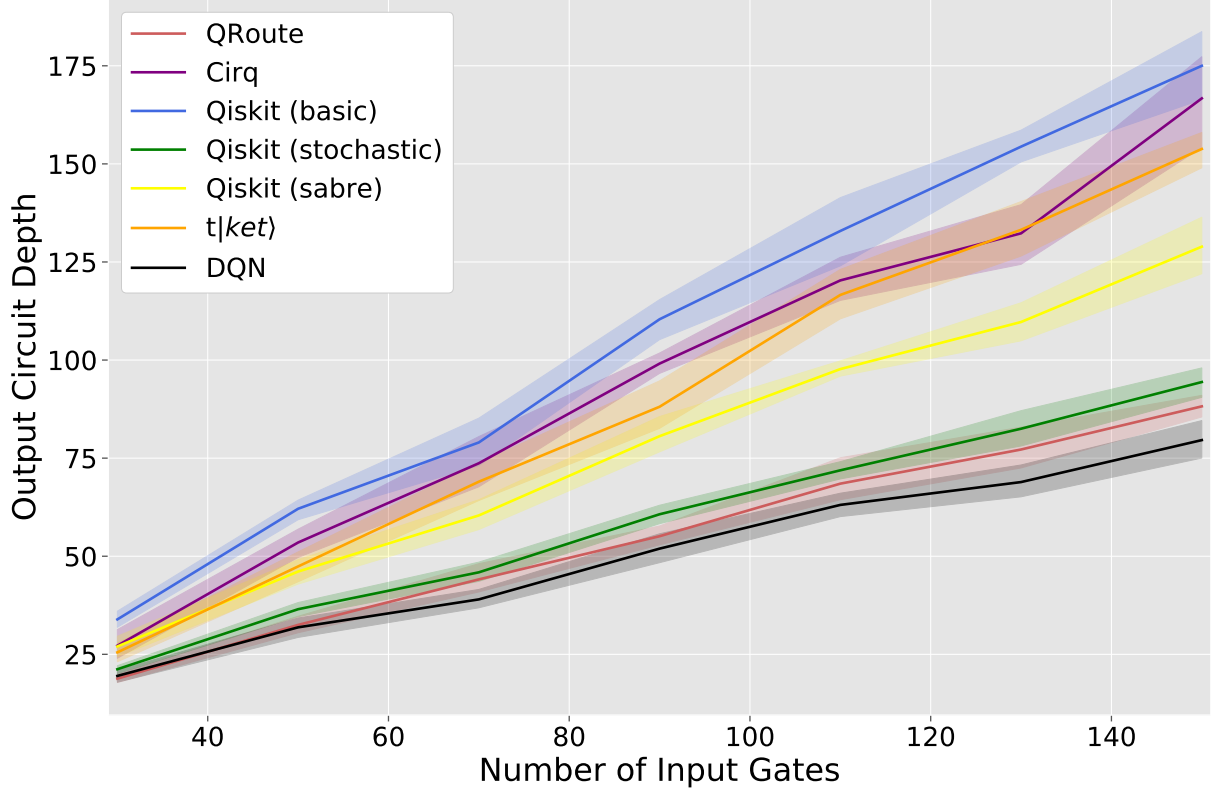
Figure 3.5: Comparative performance of routing algorithms on random circuits as a function of the number of two-qubit operations in the circuit.

3.5.1 Random Test Circuits

The first benchmark for comparing our performance comprises of random circuits. These circuits are generated on the fly and initialized with the same number of qubits as there are nodes on the device. Then two-qubit gates are put between any pair of qubits chosen at random. In our simulations, the number of such gates is varied from 30 to 150 and the results for assessing performance of different frameworks are given in Fig. 3.5. The experiments were repeated 10 times on each circuit size, and final results were aggregated over this repetition.

Amongst the frameworks compared, QRoute ranks a very close second only to Deep-Q-Network guided simulated annealer (DQN). Nevertheless, QRoute still does consistently better than all the other major frameworks: Qiskit, Cirq and t|ket⟩, and it scales well when we increase the number of layers and the layer density in the input circuit. QRoute shows equivalent performance to DQN on smaller circuits, and on the larger circuits it outputs depths which are on average ≤ 4 layers more than those of DQN. Some part of this can be attributed to MCTS, in it's limited depth search, choosing the worse of two moves with very close Q-values, resulting in the scheduling of some unnecessary SWAP operations.
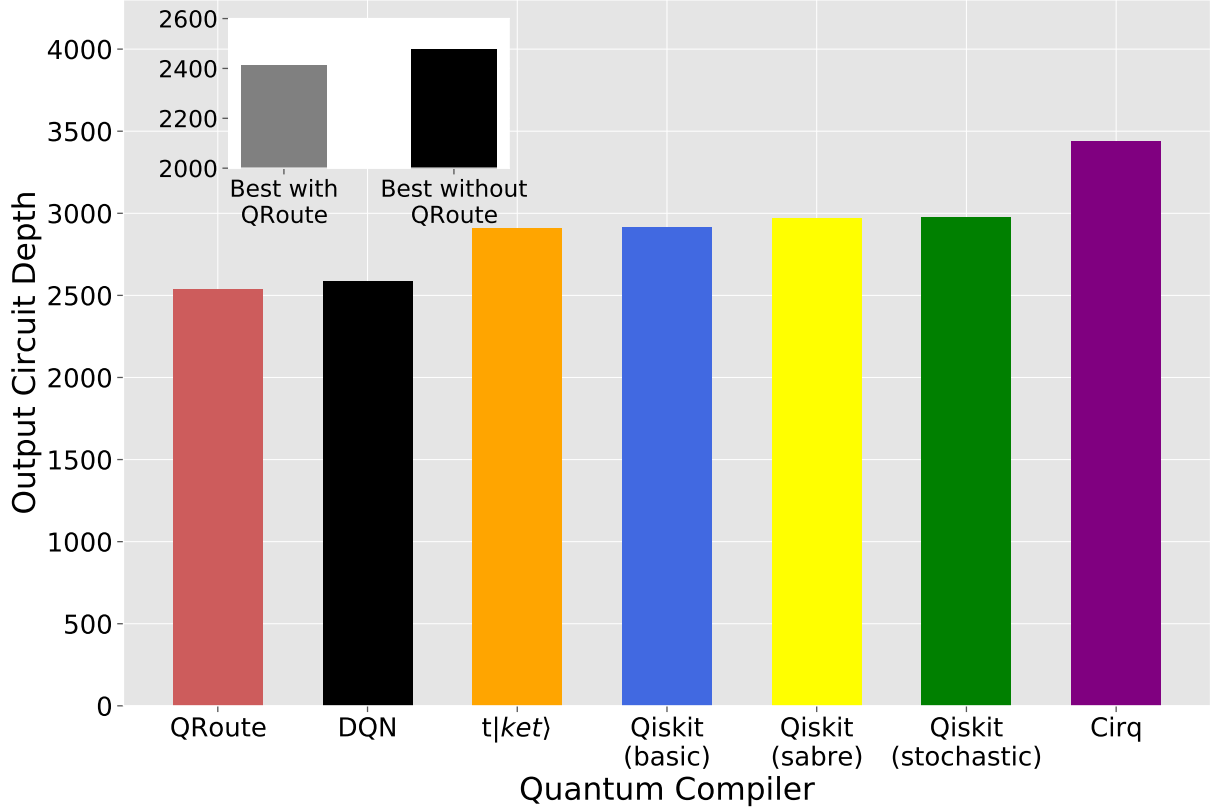
Figure 3.6: Plots of output circuit depths of routing algorithms over small realistic circuits (≤ 100 gates), summed over the entire dataset. The inset shows the results on the same data comparing the best performant scheduler excluding and including QRoute on each circuit respectively.

3.5.2 Small Realistic Circuits

Next we test on the set of all circuits which use 100 or less gates from the IBM-Q realistic quantum circuit dataset used by [65]. The comparative performance of all routing frameworks has been shown by plotting the depths of the output circuits summed over all the circuits in the test set in Fig. 3.6. Since the lack of a good initial qubit allocation becomes a significant problem for all pure routing algorithms on small circuits, we have benchmarked QRoute on this dataset from three trials with different initial allocations.

The model presented herein has the best performance on this dataset. We also compare the best result from a pool of all routers including QRoute against that of another pool of the same routers but excluding QRoute. The pool including QRoute gives on average 2.5% lower circuit depth, indicating that there is a significant number of circuits where QRoute is the best routing method available.

On this dataset also, closest to QRoute performance is shown by Deep-Q-Network guided simulated annealer. To compare performances, we look at the average circuit depth ratio (CDR), which is defined
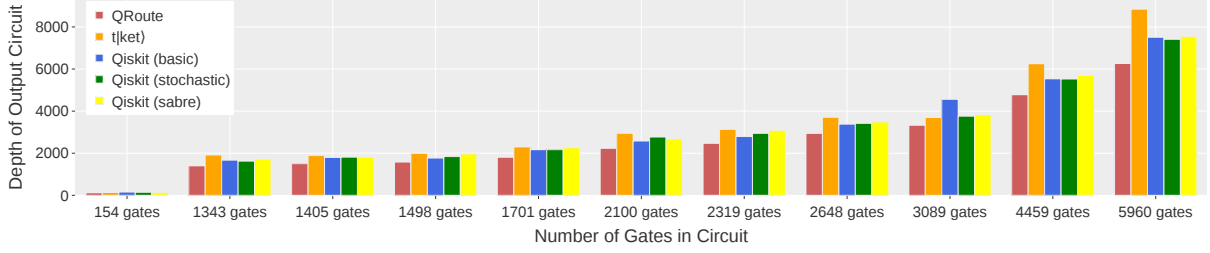
Figure 3.7: The results over eight circuits sampled from the large realistic dataset benchmark, the outputs of each routing algorithm are shown for every circuit.

by [38]:

$$\text{CDR} = \frac{1}{\#\text{circuits}} \sum_{\text{circuits}} \frac{\text{Output Circuit Depth}}{\text{Input Circuit Depth}} \tag{3.7}$$

The resultant CDR for QRoute is 1.178, where as the reported CDR for the DQN is 1.19. In fact, QRoute outperforms DQN on at least 80% of the circuits. This is significant because in contrast to the random circuit benchmark, the realistic benchmarks consist of the circuits that are closer to the circuits used in useful computation.

3.5.3 Large Realistic Circuit

For final benchmark, we take eight large circuits ranging from 154 gates to 5960 gates in its input from the IBM-Quantum realistic test dataset [65]. The results are plotted in Fig. 3.7. QRoute has the best performance of all available routing methods: Qiskit and t|ket⟩, on every one of these sampled circuits with on an average 13.6% lower circuit depth, and notable increase in winning difference on the larger circuits.

The results from DQN and Cirq are not available for these benchmarks as they are not designed to scale to such huge circuits. In case of DQN, the CDR data results were not provided for the circuits over 200 gates, mainly because simulated annealing used in it is computationally expensive. Similarly, for Cirq, it takes several days to compile each of the near 5000 qubit circuits. In contrast, QRoute is able to compile these circuits in at most 4 hours, and its compilation process can be sped up by reducing the depth of the search. Spending more time, however, helps MCTS to better approximate the Q-values leading to circuits with lower resulting depth.

3.6   Discussion and Conclusion

In this article, we have shown that the problem of qubit routing has a very powerful and elegant formulation in Reinforcement Learning (RL) which can surpass the results of any classical heuristic algorithm across all sizes of circuits and types of architectures. Furthermore, the central idea of building up solutions step-by-step when searching in combinatorial action spaces and enforcing constraints using

mutex locks, can be adapted for several other combinatorial optimization problems [28, 60, 61, 1, 23]. Our approach is flexible enough to compile circuits of any size onto any device, from small ones like IBMQX20 with 20 qubits, to much larger hardware like Google Sycamore (results provided in supplementary) with 53 qubits (the Circuit Depth Ratio for small realistic circuits on Google Sycamore was 1.64). Also, it intrinsically deals with hardware having different primitive instruction set, for example on hardware where SWAP gates are not a primitive and they get decomposed to 3 operations. QRoute enjoys significant tunability; hyperparameters can be changed easily to alter the tradeoff between time taken and optimality of decisions, exploration and exploitation, etc.

QRoute is a reasonably fast method, taking well under 10 minutes to route a circuit with under 100 operations, and at most 4 hours for those with upto 5000 operations, when tested on a personal machine with an i3 processor (3.7 GHz) and no GPU acceleration. Yet more can be desired in terms of speed. However, it is hard to achieve any significant improvement without reducing the number of search iterations and trading off a bit of performance. More predictive neural networks can help squeeze in better speeds.

One of the challenges of methods like DQN, that use Simulated Annealing to build up their actions is that the algorithm cannot plan for the gates which are not yet waiting to be scheduled, those which will come to the head of the list once the gates which are currently waiting are executed [38]. QRoute also shares this deficiency, but the effect of this issue is mitigated by the explicit tree search which takes into account the rewards that will be accrued in the longer-term future. There is scope to further improve this by feeding the entire list of future targets directly into our neural network by using transformer encoders to handle the arbitrary length sequence data. This and other aspects of neural network design will be a primary facet of future explorations. Another means of improving the performance would be to introduce new actions by incorporating use of BRIDGE gates [20] and gate commutation rules [15] alongside currently used SWAP gates. The advantage of former is that it allows running CNOT gates on non-adjacent qubit without permuting the ordering of the logical qubits; whereas, the latter would allow MCTS to recognize the redundancy in action space, making its exploration and selection more efficient.

Finally, we provide an open-sourced access to our software library. It will allow researchers and developers to implement variants of our methods with minimal effort. We hope that this will aid future research in quantum circuit transformations. For review we are providing, the codebase and a multimedia in the supplementary.

On the whole, the Monte Carlo Tree Search for building up solutions in combinatorial action spaces has exceeded the current state of art methods that perform qubit routing. Despite its success, we note that QRoute is a primitive implementation of our ideas, and there is great scope of improvement in future.

Chapter 4

qLEET: Visualizing Loss Landscapes, Expressibility, Entangling power and Training Trajectories for Parameterized Quantum Circuits

4.1 Abstract

We present qLEET, an open-source Python package for studying parameterized quantum circuits (PQCs), which are widely used in various variational quantum algorithms (VQAs) and quantum machine learning (QML) algorithms. qLEET enables computation of properties such as expressibility and entangling power of a PQC by studying its entanglement spectrum and the distribution of parameterized states produced by it. Furthermore, it allows users to visualize the training trajectories of PQCs along with high-dimensional loss landscapes generated by them for different objective functions. It supports quantum circuits and noise models built using popular quantum computing libraries such as Qiskit, Cirq, and Pyquil. In our work, we demonstrate how qLEET provides opportunities to design and improve hybrid quantum-classical algorithms by utilizing intuitive insights from the ansatz capability and structure of the loss landscape.

4.2 Introduction

Parameterized quantum circuits (PQCs) are one of the fundamental components of variational quantum algorithms (VQAs). They are responsible for evolving the qubits system to a state dependent on the series of parameters $(\vec{\theta})$ provided by a classical processor and the objective function where the initial state of the qubit system might be the ground state $|0\ldots0\rangle$, or some other state $|\psi_0\rangle$ of a particular form. The PQC $(U(\vec{\theta}))$ is also popularly referred to as ansatz, as we have already seen in the previous two chapters. Their structure dramatically affects the performance of VQAs as they influence both (i) convergence speed, i.e., the number of quantum-classical feedback iterations, and (ii) closeness of the final state $(|\psi(\vec{\theta})\rangle)$ to a state that optimally solves the problem $(|\psi(\vec{\theta^*})\rangle)$, i.e., the overlap or the fidelity $(\mathcal{F} = |\langle\psi(\vec{\theta})|\psi(\vec{\theta^*})\rangle|^2)$ between the final state and the target state.

Therefore, it becomes imperative to be able to design PQCs for a given problem. However, this is not straightforward because their design depends not only on the problem itself but also on the quantum hardware that executes them. After all, some essential properties like depth of circuit post compilation depend on the hardware's topology and the supported basis gates. Nevertheless, based on this, there exist three main classes of ansätze: (i) problem-inspired ansatz, where the evolutions of generators derived from properties of the given system are used to construct the PQCs, (ii) hardware-efficient ansatz, where a minimal set of quantum gates native to a given device are used to construct the PQCs, (iii) adaptive ansatz, which is midway between the former two ansätze. Using these three classes, one can come up with numerous ansatz designs for any given problem, but to finally choose one, we need to compare them.

The primary motivation behind the development of qLEET[1] [5] stems from this need to have a framework for analyzing the capabilities of parameterized quantum circuits. It does so by allowing users to study various properties related to the behavior of PQCs and assess their trainability. In particular, it will enable visualization of the loss landscape of a PQC for a given objective function and its training trajectory in the parameter space. Furthermore, it allows calculation of some essential properties of PQCs, such as their expressibility and entangling capabilities [49]. It is integrable with other popular libraries such as Qiskit [2], Cirq [10], or PyQuil [51] and also supports instruction-set languages like OPENQasm [12] and Quil [51].
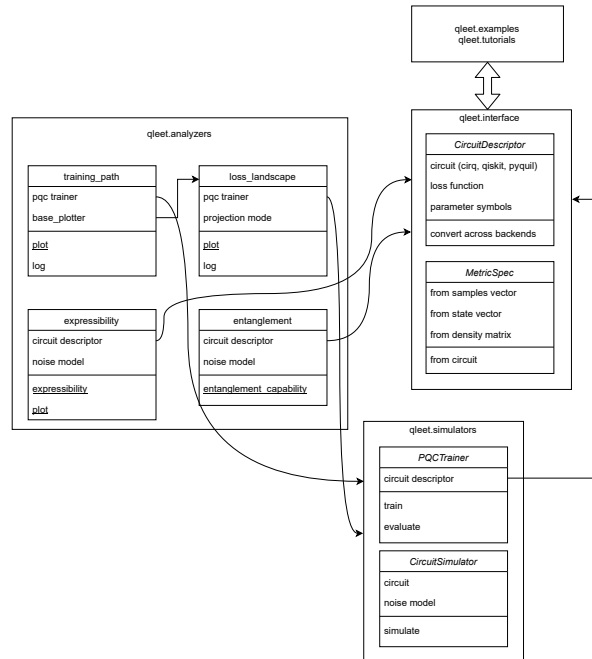
Figure 4.1: Architecture stack for qLEET

(a) p = 1

(b) p = 4

(c) p = 8
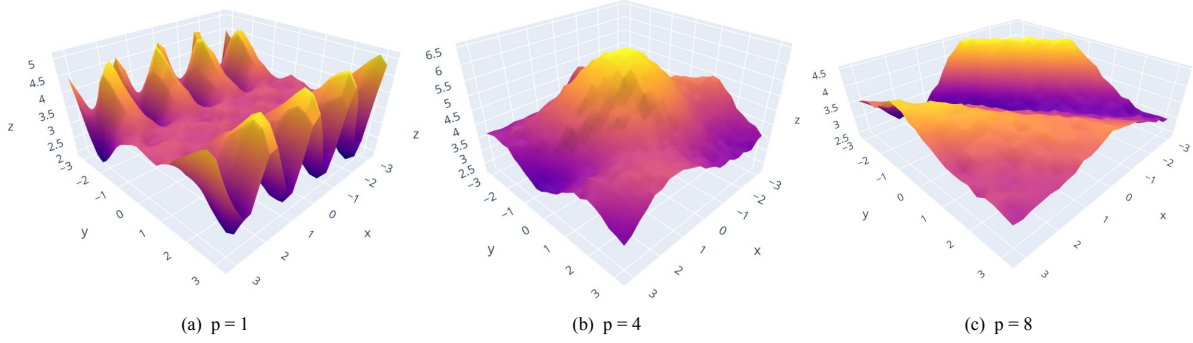
Figure 4.2: Loss landscapes for solving Max-Cut problem for an Erdos-Renyi graph with 12 nodes and 0.5 edge probability, using QAOA for different values of p.

4.3 Overview

We implement qLEET by following a modular approach where similar functionalities are grouped under one single module that can interact with one another, as shown in Fig. 4.1. In total, there exist the following three modules:

1. **Interface module**: This allows building either PQC or the workflow of the variational computation by also specifying a set of parameters, objective or cost function, and metrics based on the final state to which the circuit evolves. It also provides a circuit wrapper called `CircuitDescriptor` that makes the computation hardware agnostic.

2. **Simulators module**: This module helps train in setting up the training routine (`PQCTrainer`) for the variational computations and simulations routine (`CircuitSimulator`) for standalone PQCs with or without noise depending upon whether the user provides a noise model or not.

3. **Analyzers module**: This module takes in a `CircuitDescriptor` or `CircuitSimulator` object to compute various properties such as loss landscape or training trajectory in the case of a variational computation and expressibility or entangling capacity value for standalone PQCs.

In addition to these, we provide add-ons in the form of tutorials and predefined examples. Furthermore, we list below the theory for a range of features that are supported by qLEET.

4.4 Trainability of PQCs

Let us consider a PQC $\hat{U}(\vec{\theta})$ and an objective function $\mathcal{C}$. The process of training of $U(\vec{\theta})$ with respect to $\mathcal{C}$ is defined as:

$$\mathcal{C} = \min \mathrm{Tr}[O\rho(\vec{\theta_t})] \tag{4.1}$$

$$\vec{\theta}_{t+1} = \vec{\theta}_t - \gamma \nabla_{\vec{\theta}} \mathcal{C} \tag{4.2}$$

Here, $\rho(\vec{\theta})$ is an arbitrary quantum state produced by the PQC, and $\hat{O}$ is a Pauli observable. This Pauli observable can either be a physical Hamiltonian (as in the case of molecular VQE) or any other Hermitian observable. To assess the trainability of PQC, we would require to calculate the $\mathcal{C}$ and also the contribution of a parameter θ_v to $\nabla_{\vec{\theta}} \mathcal{C}$, i.e., $\partial \mathcal{C}/\partial \theta_v$. For a majority of values, these contributions will be unbiased, non-vanishing, and non-exploding, the PQCs can be trained successfully for the given $\mathcal{C}$. These could be better understood by visualizing the loss landscape and training path.

4.4.1 Loss Landscape

Loss landscape is a visual representation of the loss value of different areas of our parameter space, of how different parameter vectors result in states with different values of the objective . We perform this analysis typically around the point of convergence θ^* to visualize the smoothness of loss surface and identify features like local minima, barren plateaus, ridges and valleys, etc. [24]

To plot the loss landscape, we compute the loss function value for all the parameters in the orthonormalized 2-D subspace S with basis vectors θ_i sampled from the whole parameter space as detailed in equation 4.3. Based on how this sampling is performed, we can gather different information about the landscape. For example, suppose we use a principal component analysis (PCA) over the set of parameters at each training step. In that case, we get the vectors, i.e., the directions in space for which major movements occurred for our PQC during training. Similarly, other methods for obtaining subspace could be used, such as doing random sampling of basis vectors or t-SNE (t-Distributed Stochastic Neighbor Embedding) of the parameter vectors encountered in the training trajectory. We gain some crucial insights from the loss landscape, such as the roughness, flatness, and the presence of repeating patterns, which could help one adapt our training strategy by tweaking the optimization routine, evaluation metric, etc.

$$
\begin{aligned}
f(\alpha_i) &= \mathcal{C}_{\text{PQC}}(\theta^* + \sum_i \alpha_i \theta_i) \\
&= \sum_O \text{Tr}\left[O \rho\left(\theta^* + \sum_i \vec{\alpha_i \theta_i} \right) \right]
\end{aligned}
\tag{4.3}
$$

4.4.2 Training Trajectory

In addition to the loss landscape, it is also essential to visualize the training paths for PQC. Plotting these training trajectories over several re-initializations helps us learn about convergence properties of the parametrized quantum circuits and their optimization schedules. We plot the entire set of parameter vectors over all the re-initializations collected from each timestep of the training process θ_k^t. Similar to the loss landscapes visualization, we project these parameter vectors down to a 2-dimentional subspace which can be chosen randomly, via PCA of encountered parameter vectors, or t-SNE of the same. The
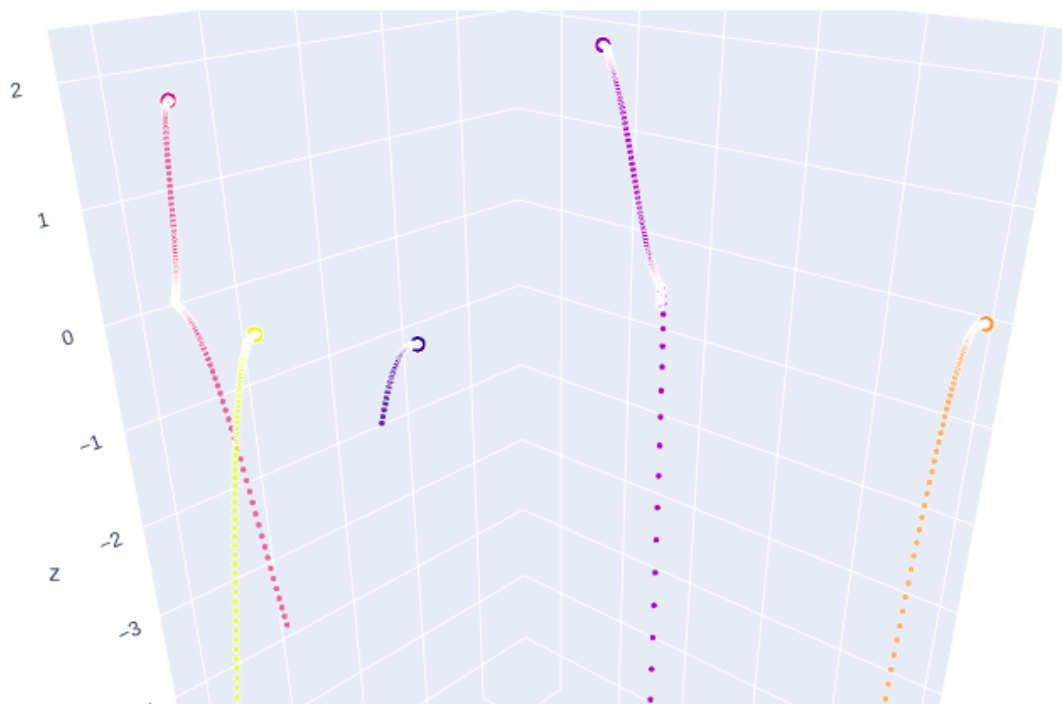
Figure 4.3: This shows a 2-D projection of the parameter vectors plotted on the X-Y axes for 5 different re-initializations, each shown using a different color. These are collected over the entire optimization run and the final value of each run is plotted with a larger blob, visible at the top of each trajectory. The z-axis shows the loss value at that parameter vector. In this plot, it is visible that the trajectories do not mix and instead ascend up their own local optima, which indicates that sufficient exploration of the loss landscape was not performed and the optimization was very much local in nature and vary over different runs.
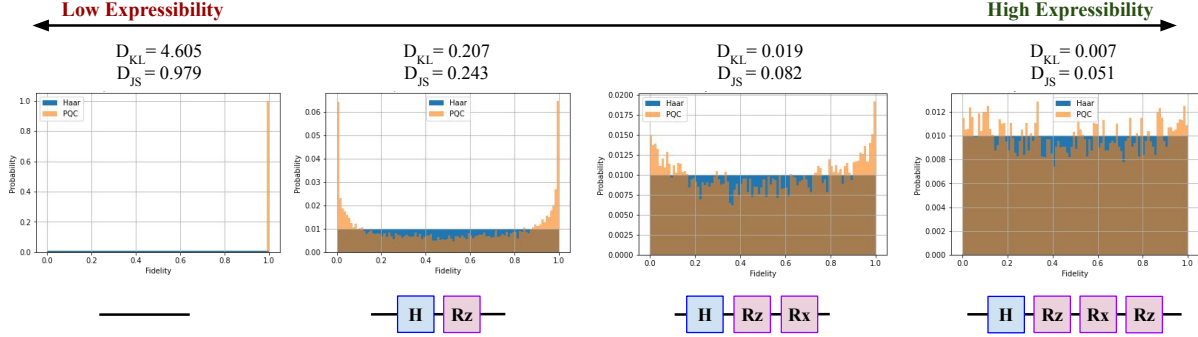
Figure 4.4: Quantifying expressibility for single-qubit circuits. For each of the four circuits show here, 1000 sample pairs of circuit parameter vectors were uniformly drawn, corresponding to 2000 parameterized states. Histograms of estimated fidelities (orange) are shown, overlaid with fidelities of the Haar-distributed ensemble (blue), with the computed Kullback-Leibler (KL) and Jensen-Shannon Distance (JS) divergences reported above the histograms.

2-D projections of the parameter trajectories can also be plotted on the loss surface, with the loss values on the 3rd axis. [26]

4.4.3 Expressibility

Sampling states $|\psi(\vec{\theta})\rangle$ from a PQC, $\hat{U}(\vec{\theta})$, for a randomly sampled parameter vector $\vec{\theta}$ generates a distribution of states. The deviation of this distribution from the Haar measure is defined as *Expressibility*, where the Haar measure samples the full Hilbert space uniformly.

$$A^{(t)} = \left\| \int_{\text{Haar}} (|\psi\rangle \langle\psi|)^{\otimes t} d\psi - \int_{\vec{\theta}} (|\psi(\vec{\theta})\rangle (\langle\psi(\vec{\theta})|)^{\otimes t} d\psi(\vec{\theta}) \right\|_{\text{HS}}^2 \tag{4.4}$$

where $\int_{\text{Haar}} d\psi$ denotes the integration over a state $|\psi\rangle$ distributed according to the Haar measure and $\|A\|_{\text{HS}}^2 = \text{Tr}(A^\dagger A)$ the Hilbert-Schmidt norm. As shown in [49], we can compute the quantity in Eq. 4.4 as the divergence between the resulting distribution of state fidelities generated by the sampled ensemble of parameterized states to that of the ensemble of Haar random states.

$$\text{Expr} = D(\hat{P}_{PQC}(F; \theta) | P_{Haar}(F)) \tag{4.5}$$

An ansatz circuit U with a small *Expr* value is more expressive because it would mean that the states generated by it match the Haar measure more closely. This is represented in Fig. 4.4, where we see the fidelity distribution of PQC and Haar measure increases as the single-qubit circuit is made more expressive by adding Pauli rotation gates. In general, when we train a PQC to represent a particular unknown target state, it is more likely for a highly expressive PQC to be able to represent the target state. Hence, Expressibility is a crucial measure to compare the effectiveness of a pair of PQCs.

4.4.4 Entangling Capability

Another important quantifier for PQC is its power to create entangled states. In general, people use different kinds of entanglement measures to capture different properties of multipartite entanglement present in the system. In [49], Meyer-Wallach Q measure [29] has been proposed to estimate the number of entangled states can be produced by a PQC, by measuring the average entanglement between individual qubits and the rest. The entangling capability of a PQC is then defined as the average, Q, of states randomly sampled from the circuit.

$$Q = \frac{2}{|\vec{\theta}|} \sum_{\theta_i \in \vec{\theta}} \left(1 - \frac{1}{n} \sum_{k=1}^{n} \mathrm{Tr}(\rho_k^2(\theta_i)) \right) \tag{4.6}$$

where ρ_k is the density matrix of the k-th qubit. Similarly, there is another measure called Scott Measure [27], which is a generalized version of the Meyer-Wallach measure. It gives m entanglement measures, each of which will measure the average entanglement between blocks of m qubits and the rest of the system. As m increases, Q_m becomes more sensitive to correlations of an increasingly global nature. The entangling capability of a PQC in this case is defined by a sequence of the following Q_m measures:

$$Q_m = \frac{2^m}{(2^m - 1)|\vec{\theta}|} \sum_{\theta_i \in \vec{\theta}} \left(1 - \frac{m!(n-m)!}{n!} \sum_{|S|=m} \mathrm{Tr}(\rho_S^2(\theta_i)) \right)$$
$$m = 1, \ldots, \lfloor n/2 \rfloor \tag{4.7}$$

4.4.5 Entanglement Spectrum

In the previous subsection, we assessed the entangling power of an ansatz using two entanglement measures. However, to fully characterize the various properties of the entanglement produced by an ansatz, one must make use of the entanglement spectrum [62, 59], which is defined as the spectrum of eigenvalues of the entanglement Hamiltonian.

$$H_{\mathrm{ent}} = -\log(\rho_A) \tag{4.8}$$

Here, $\rho_A = \mathrm{Tr}_B(\rho)$ is the reduced density matrix obtained by the typical bipartition of the N qubit system into subsystems A and B. A crucial feature of H_{ent} is that its eigenvalues ξ_k follow the Marchenko-Pastur distribution [64] for states that are sampled randomly according to Haar measure. Therefore, for a PQC, both expressibility and entangling capacity could be visualized at once by looking at the distribution of eigenvalues of $H_{\mathrm{ent}}^{\mathrm{PQC}}$, which is calculated from the states generated from the sampled set of parameters $\vec{\theta}$. In Fig. 4.5, we perform the entanglement spectrum analysis on a 16 qubit PQC, which is made of L layers comprising three rotation gates on each qubit and CNOT gates between adjacent qubits, i.e., $U(\vec{\theta}) = \prod_l^L \left(\prod_{i=0}^{15} R_x(\theta_i^1) R_z(\theta_i^2) R_x(\theta_i^3) \prod_{i=0}^{14} CX(i, i+1) \right)$.

4.5 Challenges for Variational Quantum Computation

4.5.1 Barren Plateaus

The Barren Plateaus (BP) phenomenon is one of the main restrictions for VQAs. For a given problem, BP will be exhibited for a cost function $\mathcal{C}(\vec{\theta})$ whose magnitude of partial derivatives will, on average, exponentially vanish. This essentially flattens the landscape, to traverse through which one would need exponentially large precision to resolve against finite sampling noise for determining a cost-minimizing direction. It was recently shown that the noise in NISQ-era hardware could induce BPs [57]. BPs are a problem of major concern because the exponential scaling in the needed precision due to BPs could erase a potential quantum advantage with a VQA, as its complexity would be comparable to the exponential scaling typically associated with classical algorithms. Therefore, to preserve the hope of using NISQ devices to achieve quantum advantage, one must attempt to build BP resilient VQAs.

In Fig. 4.6, we show an example of BP phenomena while comparing global $\mathcal{C}_{Global}$ and local $\mathcal{C}_{Local}$ cost functions for learning Identity gate using a very simple ansatz: $R_X(0, \theta_1) R_X(1, \theta_2) CZ(0, 1)$.

$$
\begin{aligned}
\mathcal{C}_{Global} &= \langle \psi(\vec{\theta}) | \, (I - |0 \dots 0\rangle \langle 0 \dots 0|) \, |\psi(\vec{\theta})\rangle \\
&= 1 - p_{0\dots 0}
\end{aligned}
\tag{4.9}
$$

$$
\begin{aligned}
\mathcal{C}_{Local} &= \langle \psi(\vec{\theta}) | \left(I - \frac{1}{n} \sum_j |0\rangle \langle 0|_j \right) |\psi(\vec{\theta})\rangle \\
&= 1 - \frac{1}{n} \sum_j p_{0_j}
\end{aligned}
\tag{4.10}
$$

We see how the loss landscape flattens for the $\mathcal{C}_{Global}$ and the gradients vanish exponentially as well.

4.5.2 Reachability

Reachability quantifies whether a given PQC, $\hat{U}(\vec{\theta})$, with parameters $\vec{\theta}$ is capable of representing a parameterized quantum state $|\psi(\vec{\theta})\rangle$ that minimizes the cost function $\mathcal{C}$. Mathematically it is defined as [3]:

$$
f_{\mathrm{R}} = \min_{\psi \in \mathcal{H}} \langle \psi | \mathcal{C} | \psi \rangle - \min_{\vec{\theta}} \langle \psi(\vec{\theta}) | \mathcal{C} | \psi(\vec{\theta}) \rangle,
\tag{4.11}
$$

where the first term on the right side is the minimum over all states $|\psi\rangle$ of the Hilbert space, whereas the second term is the minimum over all states that can be represented by the PQC. The reachability is equal or greater than zero $f_{\mathrm{R}} \geq 0$, with $f_{\mathrm{R}} = 0$ when the PQC can generate an optimal state $|\psi(\vec{\theta^*})\rangle$ that minimizes the objective function.

4.6   Conclusion

This chapter presents an open-source library called qLEET and demonstrates its ability to analyze various properties of parameterized quantum circuits (PQCs), such as their expressibility and entangling power. We have presented a theory of expressibility and entangling capability of a PQC based on the deviation of the distribution of parameterized states produced from the Haar measure, which samples uniformly from the entire Hilbert space. We also describe the entanglement spectrum, which allows visualizing the previous two properties at once. It also allows one to study the usage of PQCs in various variational algorithms and quantum machine learning models through its training and example modules. This involves visualizing their loss landscapes and training trajectories for different objective functions and optimizers. Finally, we discuss some critical challenges for variational quantum algorithms such as Barren Plateaus and Reachability. We conclude that qLEET will provide opportunities to design new hybrid algorithms by utilizing intuitive insights from the ansatz capability and structure of the loss landscape.
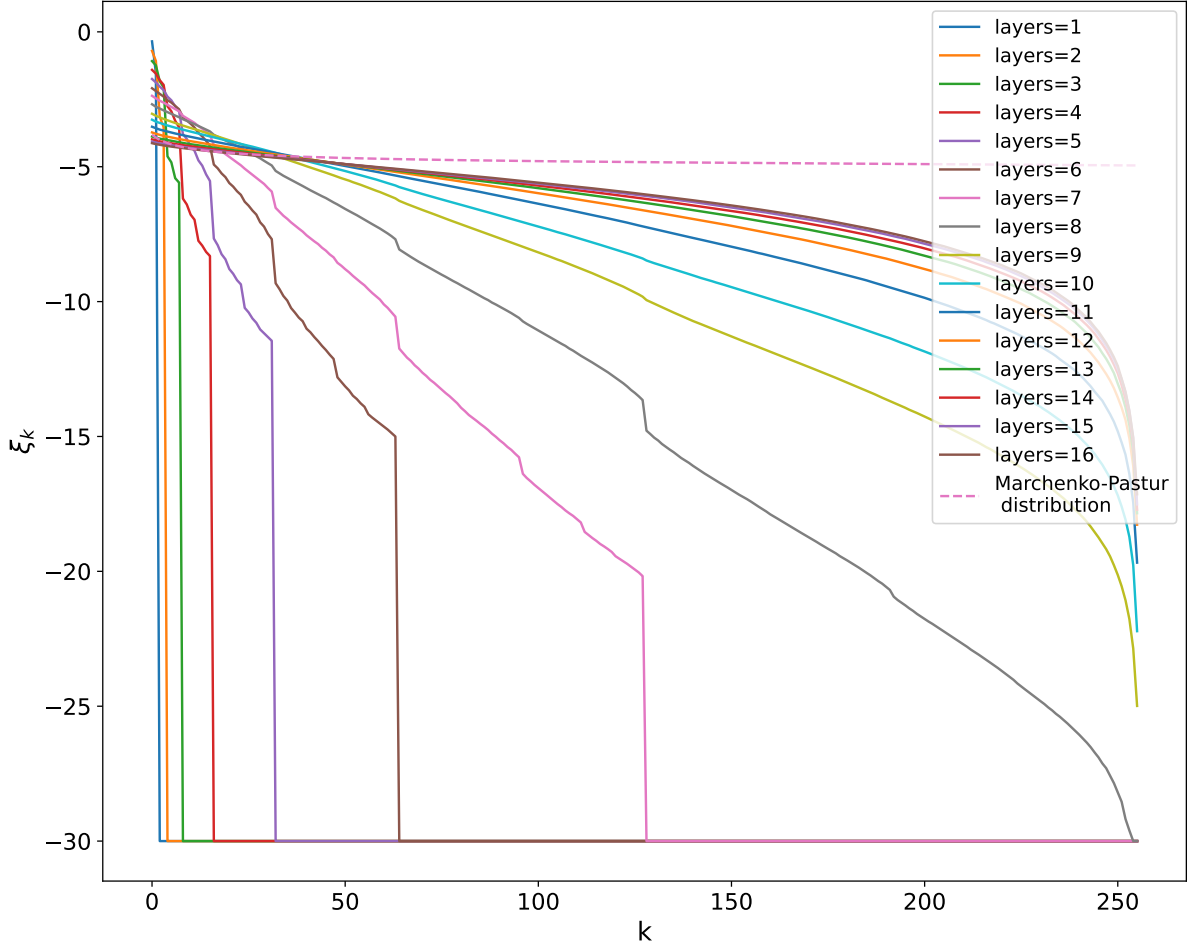
Figure 4.5: Visualizing entanglement spectrum for a PQC $U(\vec{\theta}) = \prod_l^L \left(\prod_{i=0}^{15} R_x(\theta_i^1) R_z(\theta_i^2) R_x(\theta_i^3) \ldots \prod_{i=0}^{14} CX(i, i+1) \right)$. Here, ξ_k are the eigenvalues of $H_{\text{ent}}^{U(\vec{\theta})}$ arranged in descending order and cut off at -30. The solid lines (blue to brown) represents the distribution ξ_k for different layers L and the dotted line (magenta) represents the ideal Marchenko-Pastur (MP) distribution. We see that as the number of layers is increased, the distribution of ξ_k becomes more similar to MP distribution.
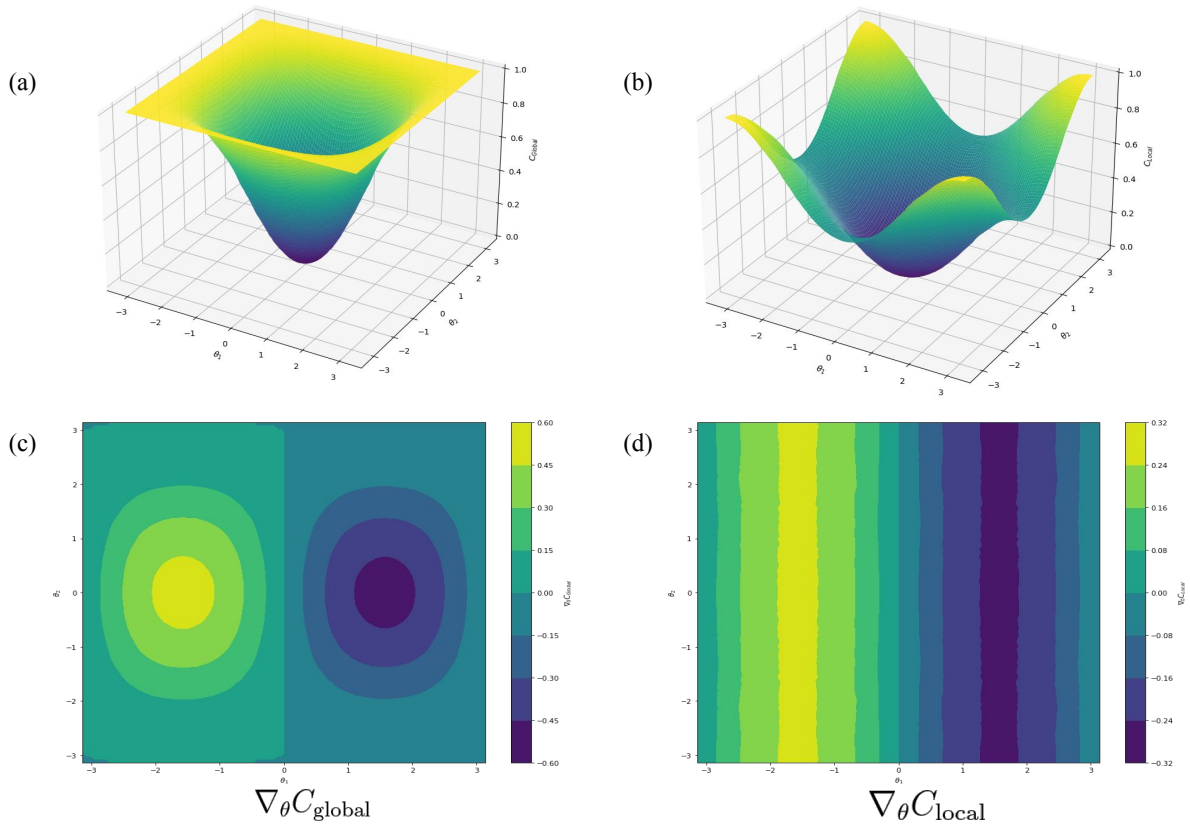
Figure 4.6: Here we show the emergence of barren plateaus in the task of learning an Identity gate using the ansatz $R_X(0, \theta_1)R_X(1, \theta_2)CZ(0, 1)$ solely based on the choice of the cost function. Figures (a) and (b) represents the loss landscape for the $\mathcal{C}_{Global}$ and local $\mathcal{C}_{Local}$ cost functions, respectively. Similarly, figures (c) and (d) represents coloured heat maps for their corresponding gradients $\nabla_\theta \mathcal{C}_{Global}$ and $\nabla_\theta \mathcal{C}_{Local}$. We see that for $\mathcal{C}_{Global}$, the gradients vanish rapidly towards the boundaries of the loss landscape.

Chapter 5

Conclusions

This dissertation shows that Quantum-inspired Machine-Learning methods can be naturally applied in various parts of the Quantum processing pipeline. We present two novel results: qRoute - a reinforcement-learning-based quantum circuit routing method, and qLEET - a variational quantum circuit visualization and evaluation framework.

In qRoute, we have shown state-of-the-art performance on various quantum circuit compilation tasks. This demonstrates that reinforcement learning settings can learn algorithms more performant than manually engineered heuristics. The routing task is one step of the long hardware-efficient algorithm development and compilation pipeline. The solution for the routing task is also a proposed solution to the general problem of decision over subsets.

In qLEET, we have shown a framework consisting of both optimization-inspired and quantum-inspired methods to evaluate the learning capabilities of variational quantum circuits. These visualized metrics include loss landscape, training trajectories, expressibility, entanglement capability, and parameter histogram of these circuits. The framework presented is provided with code compatible with Cirq, Qiskit, and PyTket quantum circuits and works with any choice of the loss function, optimizer, and noise model. Our method helps guide ansatz design and parametrization better for solving these variational problems. The software package for qRoute has been made publically available on `https://github.com/AnimeshSinha1309/qroute-router/`.

The routing process is just one part of the compilation process, where we convert a circuit that does not respect the constraints of the hardware to one that does. Converting a unitary directly to a hardware-constraint-respecting circuit and optimizing noise under a specific noise model instead of minimizing depth are goals that we need to design algorithms for, and qRoute seems like a viable solution candidate as a part of the pipeline for these processes. There is also scope for adaptation of the qRoute algorithm for these objectives. Similarly, qLEET can be extended with more visualizations and analyses of the optimization landscape and quantum resource utilization. qLEET has been explored on a small class of algorithms like QAOA and VQE, visualization of other algorithms and variation algorithm design and hyperparameter tuning using qLEET has been left as future scope of the project. The software package for qLEET has been made publically available on `https://github.com/QLemma/qleet`.

In the entirety, we believe that the contributions of this dissertation will help better the time-efficiency and noise-immunity of some algorithms on noisy near-term intermediate-scale quantum computers and will serve as some inspiration to pipelines for other tasks in near-term quantum algorithm design.

Appendix A

qRoute: Algorithm Details and Additional Results

A.1   MCTS Algorithm

The following is the detailed implementation of our MCTS procedure. In each iteration of the Monte Carlo Tree Search, we start at the root, keep selecting nodes unless we find that we have selected a node never seen before, expand it, and propagate the resultant rewards up the tree to its ancestors.

In our implementation, the value of the decay factor γ is different for the COMMIT actions from those of SWAP actions. We use a decay of 1.0 (i.e. no decay) for the non-commit actions and 0.95 for commit actions. So we only have decay in reward propagation across 2 different states, not within the construction of a single action. The function step(s, a) is a call to the environment to schedule the gates as described by the action a and evolve the state $s_t \xrightarrow{a} s_{t+1}$.

A.2   Results on Google Sycamore

Following is the plot of the average Circuit Depth ratio produced by our method on the Google Sycamore processor. Sycamore has a much larger size of 53 qubits. Our method manages to give an average depth ratio of 1.64 here, and is the best of all competing routing methods.

Data: state s_t
Initialize: root $\leftarrow (s_t,$ empty action set)
loop *n_mcts times*
  | (s, a) $\leftarrow$ root
  | **repeat**
  | | Compute **UCT** values using prior + noise
  | | **Select** *move* that maximizes UCT value
  | | **if** *(s, a).child[move] $\neq$ null* **then**
  | | | (s, a) $\leftarrow$ (s, a).child[move]
  | | **else**
  | | | **if** *move = COMMIT* **then**
  | | | | $s' \leftarrow$ **step** (s, a)
  | | | | $a' \leftarrow$ empty set
  | | | **else**
  | | | | $s' \leftarrow$ s
  | | | | $a' \leftarrow$ **insert** into a the qubit pair corresponding to the move
  | | | **end**
  | | | state.child[move] $\leftarrow (s', a')$
  | | | **store** reward[(s, a), move] $\leftarrow \mathcal{R}(s', a')$ - $\mathcal{R}(s, a)$
  | | **end**
  | **until** *last taken move was **expand***
  | reward $\leftarrow$ **evaluation** from model of (s, a)
  | **while** *(s, a) $\neq$ root* **do**
  | | p-move $\leftarrow$ move from parent of (s, a) to (s, a)
  | | (s, a) $\leftarrow$ parent in tree of (s, a)
  | | reward $\leftarrow$ reward[(s, a), p-move] + $\gamma \cdot$ reward
  | | **update** (s, a).Q-value[move] with reward
  | | **increment** (s, a).N-value[move] by 1
  | **end**
end
memorize the Q-values and N-values at the root for training the model later
(s, a) $\leftarrow$ root
repeat
  | (s, a) $\leftarrow$ child of (s, a) with maximum Q-value
until *move $\neq$ COMMIT*
return a

Algorithm 1: Monte Carlo tree search
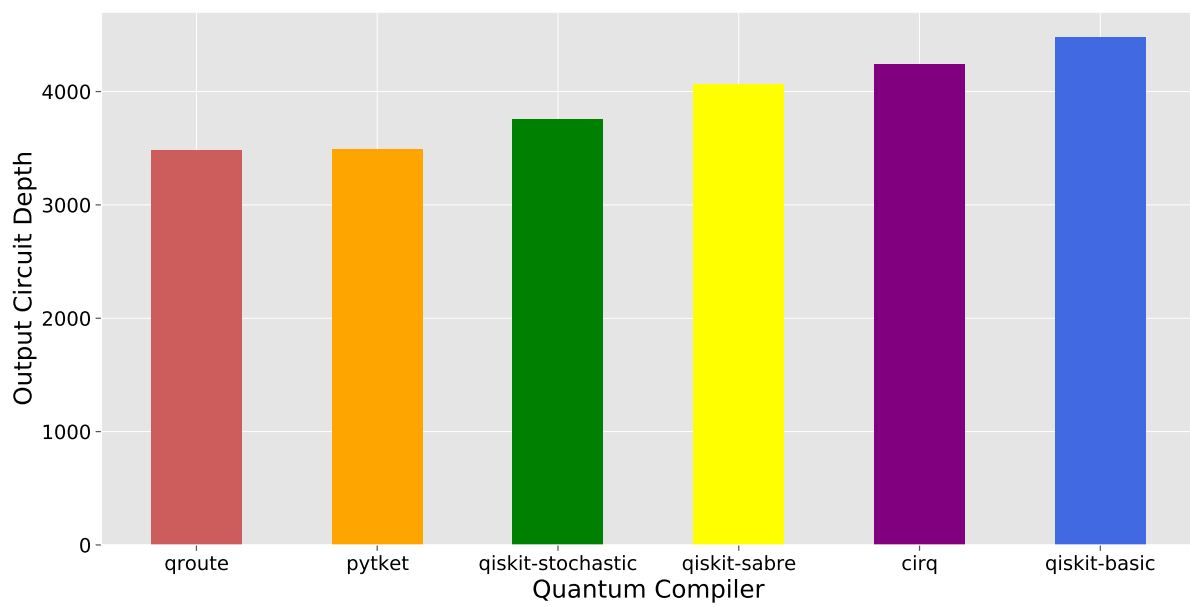
Figure A.1: A comparative of the performance of the different routing methods on the small circuit dataset when routing on Google Sycamore device.

A.3 Example of Routing Process

In this section, we show an example run of our algorithm on a 3×3 device with a normal grid topology, i.e. only qubits adjacent to each other are connected. In the images that follows, we have shown the evolution of the state, the value of the state at each timestep and the action, i.e. the set of gates which are being scheduled. Our MCTS is also responsible for constructing each action by putting together several moves (which are either adding individual gates to the action, or a committing the action for this timestep), that process is not demonstrated in the images. A point to note is that at the start of each timestep, the locks on all qubits may not necessarily be open, because there can be operations which were scheduled in a previous timestep and span over several timesteps. However, this is not the case in our example here where all the gates are assumed to take the same amount of time. We have provided a video simulation of this evolution as a supplementary, as well as the code to visualize this for other circuits.
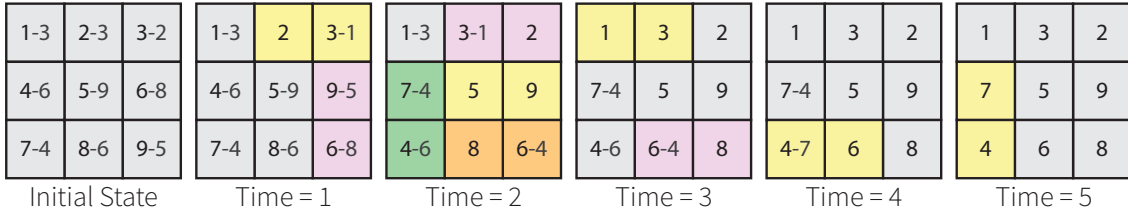
Figure A.2: The step by step evolution of the state as the circuit is getting routed. The state is shown on a 3×3 grid, where in each cell we have the node ID and the next node that it need to participate in a 2-qubit operation with. The yellow and orange colors represent that those 2 qubits have participated in a 2 qubit operation like CNOT, which was scheduled in the previous timestep. The green and purple colors represent that they have just participated in a SWAP operation. Any qubit which is colored was locked in the previous timestep when the action that scheduled it was getting constructed. At time=5, the circuit has been scheduled and none of the qubits have any targets left.

(a) Quantum circuit

(b) Decomposed circuit
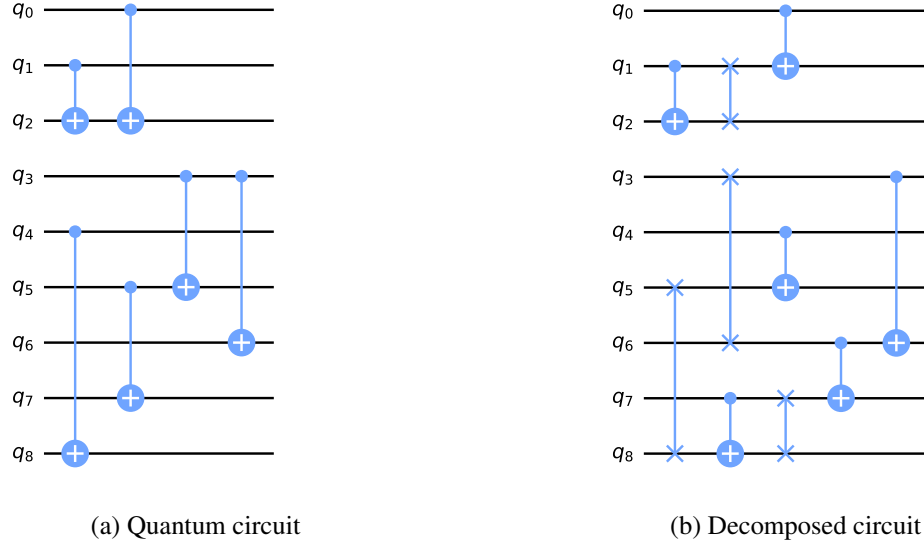
Figure A.3: This figure shows the input and output of the routing process shown above in Figure A.2. The input circuit was used to decide the targets of the qubits. Gates are added to the output circuit whenever a 2-qubit operation, whether CNOT or SWAP is applied by the router. We can check that both these circuits are equivalent

.

A.4 Tabulated Results

A.4.1 Random Test Circuits

Table A.1: Comparative results for a set of randomly generated test circuits

Input Circuit		Output Circuit Depth					
# Gates	# Layers	Qroute	Cirq	Qiskit (basic)	Qiskit (stochastic)	Qiskit (sabre)	t|ket⟩
30	11	20	29	31	21	24	19
30	11	19	36	39	23	27	28
30	10	22	32	28	23	23	23
30	8	18	24	32	20	33	26
30	7	17	19	35	17	23	30
30	11	18	39	34	22	31	26
30	10	19	22	34	21	20	26
30	9	17	24	31	21	31	33
30	10	17	24	36	22	31	21
30	10	21	23	39	22	27	23
50	18	31	57	66	41	50	46

Input Circuit		Output Circuit Depth					
# Gates	# Layers	Qroute	Cirq	Qiskit (basic)	Qiskit (stochastic)	Qiskit (sabre)	t|ket⟩
50	17	29	54	63	37	48	46
50	12	34	47	62	31	50	53
50	17	28	62	67	38	38	58
50	17	33	42	61	39	50	51
50	15	40	49	61	38	48	41
50	18	35	60	66	38	52	50
50	18	33	54	53	35	42	52
50	13	32	58	63	31	39	35
50	16	30	52	59	37	44	42
70	19	39	85	93	45	59	76
70	21	47	96	71	41	56	60
70	18	46	64	81	43	57	67
70	21	59	83	84	53	58	79
70	18	47	67	60	44	55	80
70	21	45	77	83	46	69	59
70	19	41	63	76	44	52	74
70	17	40	68	67	42	62	63
70	23	37	70	84	52	63	72
70	23	40	64	91	49	73	60
90	29	53	106	93	64	80	114
90	26	64	103	117	64	73	77
90	28	56	93	111	64	85	89
90	22	57	94	114	54	75	87
90	32	58	99	108	66	87	84
90	23	54	104	127	60	97	90
90	28	52	96	103	60	80	92
90	25	50	97	113	60	76	75
90	23	51	103	107	54	74	82
90	25	56	96	111	61	79	91
110	34	63	133	113	72	97	137
110	27	65	128	143	70	95	118
110	31	64	117	128	69	95	126
110	30	73	116	139	66	104	106
110	32	62	108	129	75	100	124

Input Circuit		Output Circuit Depth					
# Gates	# Layers	Qroute	Cirq	Qiskit (basic)	Qiskit (stochastic)	Qiskit (sabre)	t|ket⟩
110	36	68	112	135	78	93	107
110	33	94	135	158	74	99	103
110	33	67	124	110	75	96	117
110	31	64	114	129	71	101	113
110	30	65	116	145	69	97	115
130	33	74	151	149	74	113	154
130	33	91	135	166	79	122	126
130	38	77	130	162	91	123	133
130	32	77	112	153	75	116	139
130	38	71	145	151	94	113	137
130	34	66	127	153	79	98	122
130	35	75	131	151	89	101	144
130	31	70	114	157	74	107	135
130	33	76	130	141	79	102	128
130	41	95	148	161	91	102	114
150	35	87	175	151	86	109	142
150	44	92	194	195	104	154	158
150	38	84	162	177	93	136	149
150	35	79	128	178	84	123	149
150	48	96	177	195	101	138	158
150	43	92	179	167	97	126	142
150	41	90	171	185	98	120	165
150	39	85	155	158	91	125	158
150	39	88	148	182	94	135	158
150	38	89	178	162	96	123	159

A.4.2 Small Realistic Circuits

Table A.2: Comparative results for low-depth realistic test circuits

Input Circuit		Output Circuit Depth						
Circuit Name	Layers	DQN (Estimate)	Qroute	Cirq	Qiskit (basic)	Qiskit (stochastic)	Qiskit (sabre)	t|ket⟩
4gt11_83	14	17	16	22	18	19	18	15
decod24-v0_38	23	28	30	43	23	31	32	24

Input Circuit		Output Circuit Depth						
Circuit Name	Layers	DQN (Estimate)	Qroute	Cirq	Qiskit (basic)	Qiskit (stochastic)	Qiskit (sabre)	t|ket⟩
alu-v3_34	23	28	27	39	28	28	25	28
decod24-v3_45	57	68	72	79	74	84	81	77
4gt4-v0_80	71	85	89	108	111	91	109	128
alu-v0_27	15	18	16	19	21	19	17	17
miller_11	23	28	25	23	36	36	34	35
4gt11_82	18	22	19	28	22	24	23	24
mod10_176	70	84	83	113	87	94	87	96
ex1_226	5	6	7	8	10	7	8	6
4gt5_75	33	40	38	54	40	41	46	43
ising_model_10	20	24	23	40	20	20	20	5
4gt11_84	8	10	8	13	11	11	12	8
4mod5-v0_18	31	37	34	53	39	40	40	33
alu-v4_37	16	20	20	16	23	22	25	24
qft_10	34	41	53	81	113	50	75	49
4mod5-v0_19	15	18	17	25	20	19	24	26
alu-v0_27_example	15	18	16	18	21	20	17	17
ex-1_166	9	11	13	12	14	12	11	13
4mod7-v1_96	65	78	75	91	83	97	85	88
4mod5-v1_22	10	12	12	17	13	12	12	14
4gt12-v1_89	88	105	109	163	126	135	134	125
alu-v1_29	15	18	16	21	19	20	21	19
mod5d2_64	25	30	30	45	30	38	34	31
4mod7-v0_94	66	79	77	131	83	92	92	82
4gt13_91	46	55	53	64	52	68	60	52
4mod5-v0_20	9	11	11	16	17	10	10	10
alu-v2_33	15	18	19	26	17	19	20	17
4_49_16	91	109	99	138	107	129	104	128
decod24-v2_43	22	27	25	38	26	28	33	25
4gt10-v1_81	60	72	71	113	82	80	81	75
alu-bdd_288	35	42	47	60	55	52	54	44
4mod5-v1_23	30	36	33	46	45	45	48	40
one-two-three-v2_100	29	35	35	40	41	41	40	39
rd53_138	42	50	54	70	67	69	78	59

Table A.2 continued from previous page

Input Circuit		Output Circuit Depth						
Circuit Name	Layers	DQN (Estimate)	Qroute	Cirq	Qiskit (basic)	Qiskit (stochastic)	Qiskit (sabre)	t|ket⟩
alu-v2_32	64	77	72	96	88	87	98	96
rd32_270	35	42	38	53	48	53	49	40
aj-e11_165	63	75	73	103	82	90	81	82
4gt12-v0_88	77	92	90	128	116	111	107	140
decod24-v1_41	35	42	38	55	42	50	47	43
3_17_13	17	21	24	17	26	18	24	22
4mod5-v0_19	16	20	17	27	16	21	22	13
mini_alu_305	53	64	63	113	86	81	79	81
one-two-three-v0_98	59	71	71	82	69	81	77	87
4gt13_90	50	60	54	72	56	63	56	80
4mod5-bdd_287	31	37	41	48	35	48	50	35
ham3_102	11	14	14	16	15	16	15	9
alu-v1_28	16	20	16	21	20	18	22	18
rd32-v0_66	16	20	19	20	20	20	20	14
cnt3-5_179	43	52	64	91	72	61	65	83
4gt13_92	26	31	30	41	33	38	36	29
alu-v4_36	47	56	55	59	65	68	61	60
rd32-v1_68	16	20	17	21	20	20	20	14
4gt13-v1_93	27	33	29	34	35	36	36	31
4gt5_76	42	50	47	69	53	52	53	53
mod5d1_63	11	14	14	17	12	12	14	14
graycode6_47	5	6	5	9	5	5	5	5
xor5_254	5	6	5	8	10	8	8	6
decod24-bdd_294	31	37	34	50	40	40	46	37
alu-v0_26	35	42	41	62	47	48	45	54
mod5mils_65	16	20	19	21	17	21	25	18
alu-v3_35	16	20	20	25	23	21	21	24
one-two-three-v1_99	56	67	60	87	70	84	70	95
one-two-three-v3_101	29	35	34	43	36	37	38	46
4gt5_77	51	61	61	78	63	66	70	66

A.4.3 Large Realistic Circuits

Table A.3: Comparative results for long-depth realistic test circuits

Input Circuit		Output Circuit Depth				
Circuit Name	Number of Gates	Qroute	t|ket⟩	Qiskit (basic)	Qiskit (stochastic)	Qiskit (sabre)
rd84_142	154	120	154	142	138	133
adr4_197	1498	1580	1770	1840	1968	1988
radd_250	1405	1504	1799	1812	1815	1888
z4_268	1343	1400	1670	1623	1718	1914
sym6_145	1701	1806	2167	2168	2261	2299
misex1_241	2100	2231	2580	2770	2681	2944
rd73_252	2319	2468	2793	2943	3071	3132
cycle10_2_110	2648	2941	3380	3418	3485	3705
square_root_7	3089	3327	4560	3759	3822	3695
sqn_258	4459	4779	5535	5526	5696	6252
rd84_253	5960	6264	7507	7411	7537	8843

Related Publications

1. Animesh Sinha, Utkarsh Azad, and Harjinder Singh. "Qubit Routing using Graph Neural Network aided Monte Carlo Tree Search." *Proceedings of the AAAI Conference on Artificial Intelligence 2022*, *arXiv e-prints (2021): arXiv-2104*.

2. Animesh Sinha, Utkarsh Azad, and Harjinder Singh. "qLEET: Visualizing Loss Landscapes, Expressibility, Entangling power and Training Trajectories for Parameterized Quantum Circuits.", *arXiv e-prints (2022): arXiv-2204*.

Bibliography

[1] K. Abe, Z. Xu, I. Sato, and M. Sugiyama. Solving NP-Hard Problems on Graphs with Extended AlphaGo Zero. *arXiv e-prints*, May 2019.

[2] H. Abraham and et al. Qiskit: An Open-source Framework for Quantum Computing, Jan. 2019.

[3] V. Akshay, H. Philathong, M. E. S. Morales, and J. D. Biamonte. Reachability deficits in quantum approximate optimization. *Phys. Rev. Lett.*, 124:090504, Mar 2020.

[4] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[5] U. Azad and A. Sinha. qLEET, Nov. 2021.

[6] M. Baioletti, R. Rasconi, and A. Oddi. A novel ant colony optimization strategy for the quantum circuit compilation problem. In *EvoCOP*, pages 1–16, 2021.

[7] G. Brassard, P. Høyer, M. Mosca, and A. Tapp. Quantum amplitude amplification and estimation. *Quantum Computation and Information*, page 53–74, 2002.

[8] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

[9] S. Chand, H. K. Singh, T. Ray, and M. Ryan. Rollout based heuristics for the quantum circuit compilation problem. In *2019 IEEE Congress on Evolutionary Computation (CEC)*, pages 974–981. IEEE, 2019.

[10] Cirq Developers. Cirq, Mar. 2021.

[11] A. Cowtan, S. Dilkes, R. Duncan, A. Krajenbrink, W. Simmons, and S. Sivarajah. On the Qubit Routing Problem. In *14th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2019)*, volume 135 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:32. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019.

[12] A. W. Cross, A. Javadi-Abhari, T. Alexander, N. de Beaudrap, L. S. Bishop, S. Heidel, C. A. Ryan, J. Smolin, J. M. Gambetta, and B. R. Johnson. OpenQASM 3: A broader and deeper quantum assembly language. *arXiv e-prints*, Apr. 2021.

[13] E. W. Dijkstra. *Cooperating Sequential Processes*, pages 65–138. Springer New York, New York, NY, 2002.

[14] R. P. Feynman et al. Simulating physics with computers. *Int. j. Theor. phys*, 21(6/7), 1982.

[15] J. C. Garcia-Escartin and P. Chamorro-Posada. Equivalent Quantum Circuits. *arXiv e-prints*, Oct. 2011.

[16] L. K. Grover. A fast quantum mechanical algorithm for database search, 1996.

[17] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[18] S. Herbert and A. Sengupta. Using Reinforcement Learning to find Efficient Qubit Routing Policies for Deployment in Near-term Quantum Computers. *arXiv e-prints*, Dec. 2018.

[19] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

[20] T. Itoko, R. Raymond, T. Imamichi, and A. Matsuo. Optimization of Quantum Circuit Mapping using Gate Transformation and Commutation. *arXiv e-prints*, July 2019.

[21] L. Kocsis and C. Szepesvari. Bandit based monte-carlo planning. In *ECML*, 2006.

[22] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[23] A. Laterre, Y. Fu, M. Khalil Jabri, A.-S. Cohen, D. Kas, K. Hajjar, T. S. Dahl, A. Kerkeni, and K. Beguir. Ranked Reward: Enabling Self-Play Reinforcement Learning for Combinatorial Optimization. *arXiv e-prints*, July 2018.

[24] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein. Visualizing the loss landscape of neural nets. *Advances in neural information processing systems*, 31, 2018.

[25] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[26] E. Lorch. Visualizing deep network training trajectories with pca. In *ICML Workshop on Visualization for Deep Learning*, 2016.

[27] P. J. Love, A. M. van den Brink, A. Y. Smirnov, M. H. S. Amin, M. Grajcar, E. Il'ichev, A. Izmalkov, and A. M. Zagoskin. A characterization of global entanglement. *Quantum Inf Process*, 6(3):187–195, May 2007.

[28] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev. Reinforcement Learning for Combinatorial Optimization: A Survey. *arXiv e-prints*, Mar. 2020.

[29] D. A. Meyer and N. R. Wallach. Global entanglement in multiparticle systems. *J. Math. Phys.*, 43(9):4273–4278, 2002.

[30] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[31] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[32] T. M. Moerland, J. Broekens, A. Plaat, and C. M. Jonker. Monte Carlo Tree Search for Asymmetric Trees. *arXiv e-prints*, May 2018.

[33] P. R. Montague. Reinforcement learning: an introduction, by sutton, rs and barto, ag. *Trends in cognitive sciences*, 3(9):360, 1999.

[34] R. Munos. From bandits to monte-carlo tree search: The optimistic principle applied to optimization and planning. *Found. Trends Mach. Learn.*, 7:1–129, 2014.

[35] OpenAI, :, C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. d. O. Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang. Dota 2 with large scale deep reinforcement learning, 2019.

[36] A. Paler, L. M. Sasu, A. Florea, and R. Andonie. Machine learning optimization of quantum circuit layouts, 2020.

[37] A. Paler, A. Zulehner, and R. Wille. NISQ circuit compilation is the travelling salesman problem on a torus. *Quantum Science and Technology*, 6(2):025016, mar 2021.

[38] M. G. Pozzi, S. J. Herbert, A. Sengupta, and R. D. Mullins. Using Reinforcement Learning to Perform Qubit Routing in Quantum Compilers. *arXiv e-prints*, July 2020.

[39] J. Preskill. Quantum computing in the NISQ era and beyond. *Quantum*, 2:79, Aug. 2018.

[40] P. Ramachandran, B. Zoph, and Q. V. Le. Searching for Activation Functions. *arXiv e-prints*, Oct. 2017.

[41] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[42] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[43] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[44] P. W. Shor. Introduction to quantum algorithms, 2000.

[45] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, and et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, Jan 2016.

[46] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, and et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, Jan 2016.

[47] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. P. Lillicrap, K. Simonyan, and D. Hassabis. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *arXiv e-prints*, Aug. 2017.

[48] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. P. Lillicrap, K. Simonyan, and D. Hassabis. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *arXiv e-prints*, Aug. 2017.

[49] S. Sim, P. D. Johnson, and A. Aspuru-Guzik. Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms. *Advanced Quantum Technologies*, 2(12):1900070, 2019.

[50] S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, and R. Duncan. t|ket⟩: A retargetable compiler for NISQ devices. *Quantum Science and Technology*, 6(1):014003, nov 2020.

[51] R. S. Smith, M. J. Curtis, and W. J. Zeng. A Practical Quantum Instruction Set Architecture. *arXiv e-prints*, Aug. 2016.

[52] S. S. Tannu and M. K. Qureshi. Not All Qubits Are Created Equal: A Case for Variability-Aware Policies for NISQ-Era Quantum Computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, page 987–999, New York, NY, USA, 2019. Association for Computing Machinery.

[53] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[54] U. Vazirani. A survey of quantum complexity theory. In *Proceedings of Symposia in Applied Mathematics*, volume 58, pages 193–220, 2002.

[55] D. Venturelli, M. Do, E. G. Rieffel, and J. Frank. Temporal planning for compilation of quantum approximate optimization circuits. In *IJCAI*, pages 4440–4446, 2017.

[56] O. Vinyals, I. Babuschkin, J. Chung, M. Mathieu, M. Jaderberg, W. Czarnecki, A. Dudzik, A. Huang, P. Georgiev, R. Powell, T. Ewalds, D. Horgan, M. Kroiss, I. Danihelka, J. Agapiou, J. Oh, V. Dalibard, D. Choi, L. Sifre, Y. Sulsky, S. Vezhnevets, J. Molloy, T. Cai, D. Budden, T. Paine, C. Gulcehre, Z. Wang, T. Pfaff, T. Pohlen, D. Yogatama, J. Cohen, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, C. Apps, K. Kavukcuoglu, D. Hassabis, and D. Silver. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. `https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/`, 2019.

[57] S. Wang, E. Fontana, M. Cerezo, K. Sharma, A. Sone, L. Cincio, and P. J. Coles. Noise-Induced Barren Plateaus in Variational Quantum Algorithms. *arXiv e-prints*, July 2020.

[58] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic Graph CNN for Learning on Point Clouds. *arXiv e-prints*, Jan. 2018.

[59] R. Wiersema, C. Zhou, Y. de Sereville, J. F. Carrasquilla, Y. B. Kim, and H. Yuen. Exploring entanglement and optimization within the hamiltonian variational ansatz. *PRX Quantum*, 1:020319, Dec 2020.

[60] Z. Xing and S. Tu. A graph neural network assisted monte carlo tree search approach to traveling salesman problem. *IEEE Access*, 8:108418–108428, 2020.

[61] R. Xu and K. Lieberherr. Learning Self-Game-Play Agents for Combinatorial Optimization Problems. *arXiv e-prints*, Mar. 2019.

[62] Z.-C. Yang, C. Chamon, A. Hamma, and E. R. Mucciolo. Two-component structure in the entanglement spectrum of highly excited states. *Phys. Rev. Lett.*, 115:267206, Dec 2015.

[63] X. Zhou, Y. Feng, and S. Li. A monte carlo tree search framework for quantum circuit transformation. In *Proceedings of the 39th International Conference on Computer-Aided Design*, ICCAD '20, New York, NY, USA, 2020. Association for Computing Machinery.

[64] M. Žnidarič. Entanglement of random vectors. *J. Phys. A: Math. Theor*, 40(3):F105–F111, dec 2006.

[65] A. Zulehner, A. Paler, and R. Wille. IBM Qiskit developer challenge. `https://www.ibm.com/blogs/research/2018/08/winners-qiskit-developer-challenge/`, Dec. 2018. Accessed on: 2021-03-23.

[66] A. Zulehner, A. Paler, and R. Wille. An Efficient Methodology for Mapping Quantum Circuits to the IBM QX Architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(7):1226–1236, 2019.