Quantum Circuit Optimizations using Reinforcement Learning

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.

March 15, 2022

Adviser: Prof. Harjinder Singh

To Compute

Acknowledgments

Acknowledgements goes here ...

Abstract

Quantum Computation is one of the most promising futures of computing and scientific simulations. 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 towards 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

Table Page

Chapter 1

Introduction

Chapter 2

Background in Quantum Computation and Reinforcement Learning

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, following is a diagramatic illustration of the same:

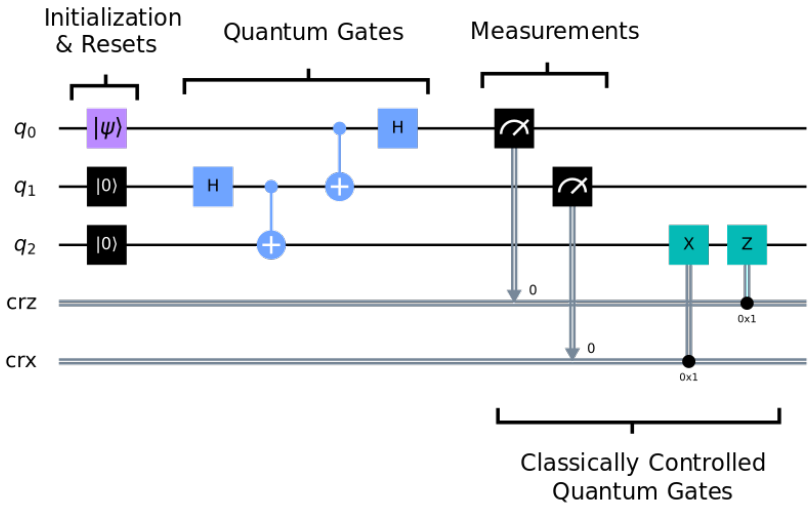


Figure 2.1: The image shows the parts of a typical quantum circuit, with 3 qubits represented by the wires and a set of gates applied to them, followed by measurement of those qubits.

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's in also has a phase term, which is represented on the bloch-sphere 2.2 on the x-y plane. The phase doesn't affect the immediate measurement of the qubit, but can affect the resultant phase and superposition when some unitary operation is applied on the qubit.

Qubits show the property of entanglement. This means that

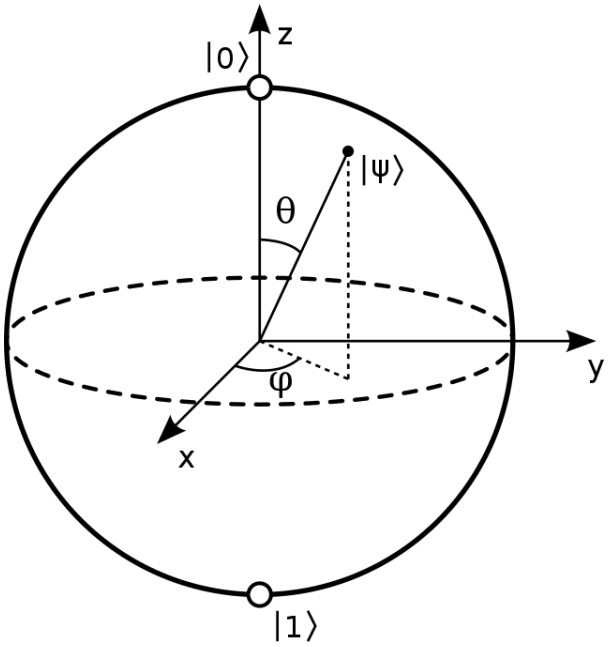


Figure 2.2: Bloch Sphere represents the state of a qubit $|\psi\rangle$. The pure states $|0\rangle$ and $|1\rangle$ are the vectors along the z-axis on the opposite poles. The angle along the x-y plane represents the phase of the qubits.

2.1.2 Variational Quantum Algorithms

2.2 Reinforcement Learning

2.2.1 What is Reinforcement Learning

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

to percieve, compute and actuate. The human mind deals with a veritable variety of problems differing greatly in their phrasing, in the solutions they admit, etc. Of this host of problem types,

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

Despite all their predictive power, these methods are limited in the set of problems they can solve. One limitation is our inability to provide a large number of labelled 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 it's 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 final result produced by the composite of steps. All these problems necessitate a machine learning method which can produce outputs over several timesteps, and be able to reason about the correctness of it's outputs based on rewards it may obtain at a different time in our process. This method is Reinforcement Learning. [29]

2.2.1.1 Markov Decision Processes

A Markov Decision Processes is any real or simulated process going on in time where each 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.

2.2.1.2 Value Function and Policy Function

At every point in time, our agent has access to the state gets to choose an action, for which it gets 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.

The agent maintains a function which is called it's **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 which each state 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 is that which leads to the maximum value function for the starting state.

Value function of one state can be written in terms of that of others, and to compute the values we can try to optimize this over all states.

$$V(s) = \sum_{a \in A} \pi(s, a) \sum_{s'} T_a(s, s')(V(s') + R_a(s, s')) \tag{2.2}$$

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

$$\begin{aligned} Q(s, a) &= \sum_{s'} T_a(s, s') \Big(R(s, s') + V(s') \Big) && (2.3) \\ &= \sum_{s'} T_a(s, s') \Big(R(s, s') + \sum_{a \in A} \pi(s', a) Q(s', a) \Big) && (2.4) \end{aligned}$$

2.2.2 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

2.2.2.1 Deep Q-Networks

The first class of models attempt 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 2.5 as:

$$Q(s, a) \leftarrow R(s, s') + \max_{a \in A} Q(s', a) \tag{2.5}$$

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 2.6 is

$$\theta_{k+1} = \theta_k - \alpha \nabla_\theta \left[\frac{1}{2} \left(Q_\theta(s,a) - \left(R(s,a,s') + \gamma \max_{a'} Q_{\theta_k}(s',a') \right) \right) \right] \Bigg|_{\theta_k} \tag{2.6}$$

Several improvements to the training efficiency and stability to the DQN algorithm have been made, a few examples are the Double DQN by [47]. These set of improvements put together have been analyzed by [15] under the name Rainbow DQN.

2.2.2.2 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 makes the choice of actions smooth and optimizable.

2.2.2.2.1 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 which 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 a policy which is inherently stochastic, where policy based methods are the better choice. One example is when designing molecules with certain properties, we want a stochastic policy so that we can sample different molecules that optimize on the target property and then rank them based on synthetic ease or the such.

3. Many a times, the action space is continuous or intractably large, and maximizing value over all 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 a un-countably infinite sized action space.

2.2.2.2.2 Method: To optimize our policy, we sample trajectories from our policy and increase the probability of actions in trajectories which high reward get increase, and those with lower reward decrease.

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 2.7 follows from a mathematically involved derivation done in [].

$$\nabla_\theta U(\theta) \leftarrow \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(u_t^{(i)} | s_t^{(i)}) \left(\sum_{k=t}^{H-1} R(s_k^{(i)}, u_k^{(i)}) - b(s_t^{(i)}) \right) \tag{2.7}$$

2.2.2.2.3 Other Nuances: Despite having the gradient that we need to update along, it's unclear what learning rate we should use to perform 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 learnt till now. Trust Region policy optimizations (TRPO) by [36] and Proximal Policy Optimizations (PPO) by [38] are methods that address this. Furthermore, to increase sample efficiency, Direct Deterministic Policy Gradients (DDPG) by [21] and Soft Actor critic (SAC) [13] are used. These methods have not seen great application in chemistry but hold great promise given their popularity in other reinforcement learning sub-domains.

2.2.2.3 Actor Critic Methods

In equation 2.7, 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 for each action now has to predict a quantity called the advantage, which represents the relative value of the actions and abstracts out the value of the state.

$$A(s, a) = Q(s, a) - V(s) \tag{2.8}$$

This is implemented in practice using two networks, an actor network, which estimates the values of the actions, and a critic network, that estimates the resultant values of the states which we subtract as 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 have been discussed in detail by [26, 27, 37]

2.2.2.4 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 tree would grow combinatorially big (molecule generation via such means

would have every possible molecule and it's substructure as a node in the tree), we use reinforcement learning to find out the most promising nodes. Monte Carlo Tree Search is one such method, which has gained prominence due to it's use in AlphaGo by [39] to play Go and in AlphaZero by [41] to play Chess, Go, and other games with no human supervision during training.

MCTS has been used in extensively 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

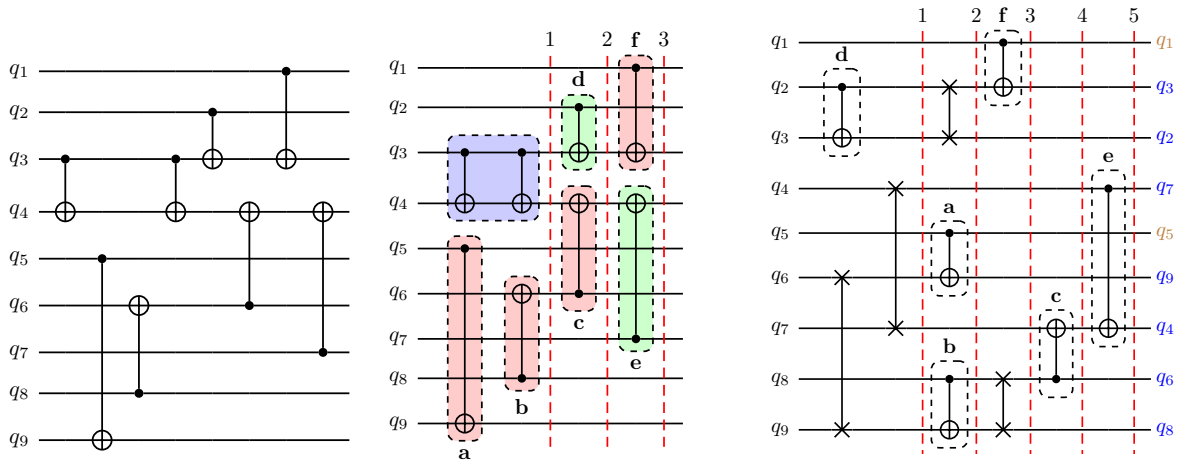
3.1 Abstract

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 [34]

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 [9]. 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 [9]. The output instructions in the transformed quantum circuit should follow

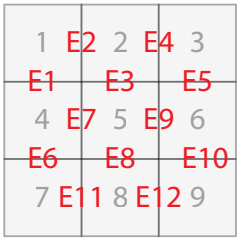


(a) Quantum circuit (b) Decomposed circuit (c) Decomposed transformed circuit

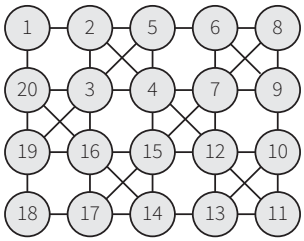
Figure 3.1: An example of qubit routing on a quantum circuit for 3×3 grid architecture (Figure 3.2a). (a) For simplicity, the original quantum circuit consists only of two-qubit gate operations. (b) Decomposition of the original quantum circuit into series of slices such that all the instructions present 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 [33].

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 [9, 14, 33]. 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 [17, 30, 18]. 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) [50], 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, [14]. 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 [12].

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:

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

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 [46]. 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 [58], 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 [9], showing equivalence to the travelling salesman problem by [32], machine learning based methods by [31], and heuristic approaches by [48], [5], [7], etc. A reinforcement learning in a combinatorial action space solution was proposed by [14], 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 [33].

Recently, Monte Carlo tree search (MCTS), a popular reinforcement learning algorithm [6] previously proven successful in a variety of domains like playing puzzle games such as Chess and Go [40], and was used by [55] 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 [55].
- 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 [14, 33]. 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.

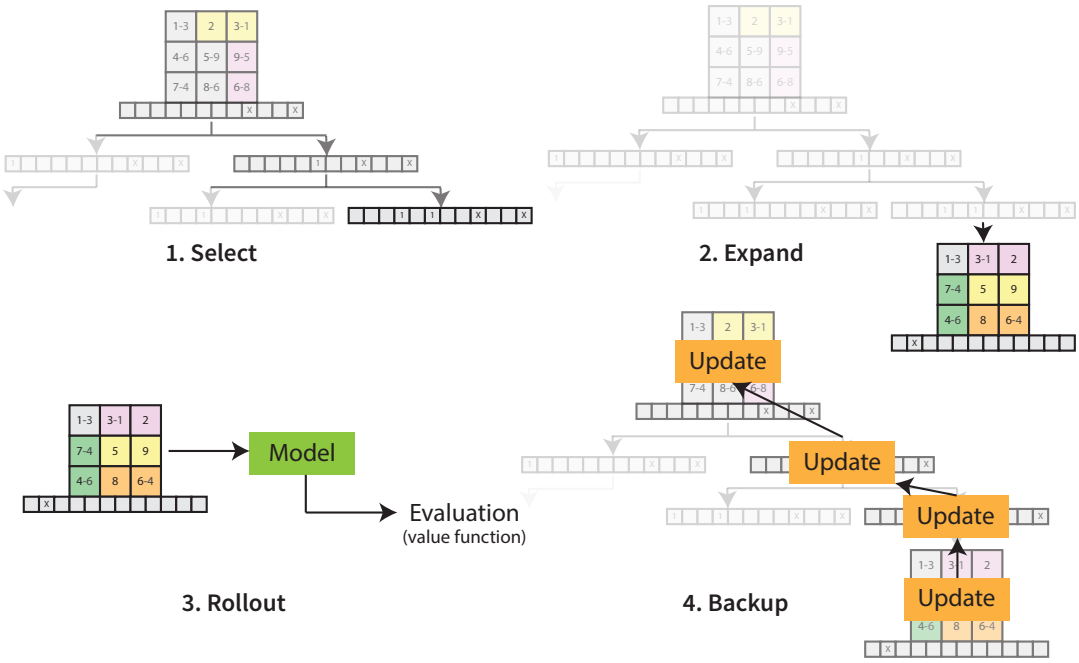


Figure 3.3: Iteration of a Monte Carlo tree search: (i) select - recursively choosing a node in the search tree for exploration using a selection criteria, (ii) expand - expanding the tree using the available actions if an unexplored leaf node is selected, (iii) rollout - estimating long term reward 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, $\mathcal{M} : Q \to N$, 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 [12]. 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" [11]. 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 [18] to balance exploration and exploitation as we traverse through the search tree. Moreover, as this problem 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 [28]:

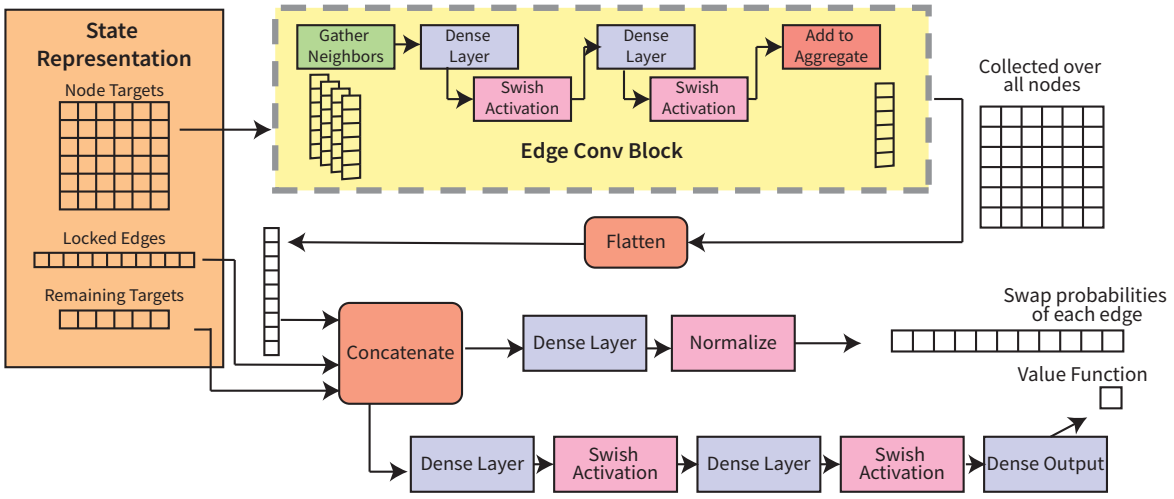


Figure 3.4: Graph neural network architecture that approximates the value function and the policy function.

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

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 [42]. 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).

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

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

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 [50], followed by some fully-connected layers with Swish [35] 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 [33], (iii) Cirq [8], and (iv) t|ket⟩ from Cambridge Quantum Computing (CQC) [44]. Qiskit's transpiler uses gate commutation rules while perform qubit routing. This strategy is shown to be advantageous in achieving lower circuit depths [16] 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 [33].

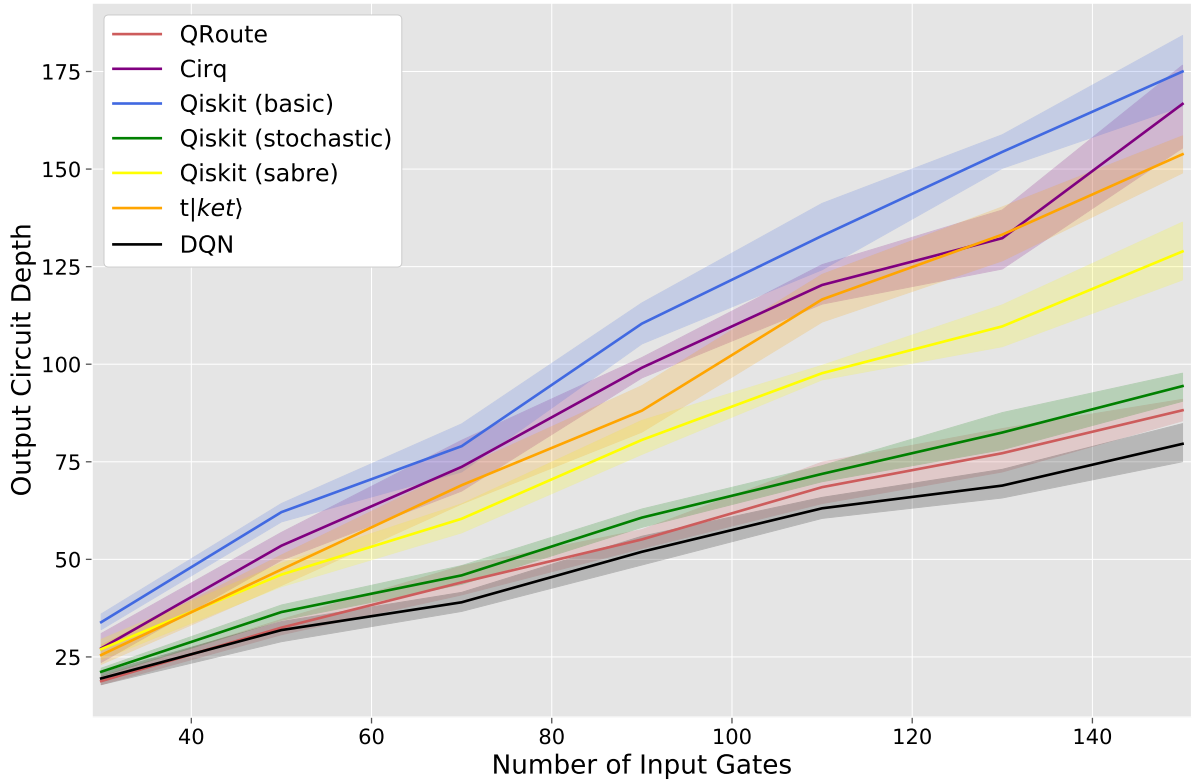


Figure 3.5: Comparative performance of routing algorithms on random circuits as a function of the number of two-qubit operations in the circuit.

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.

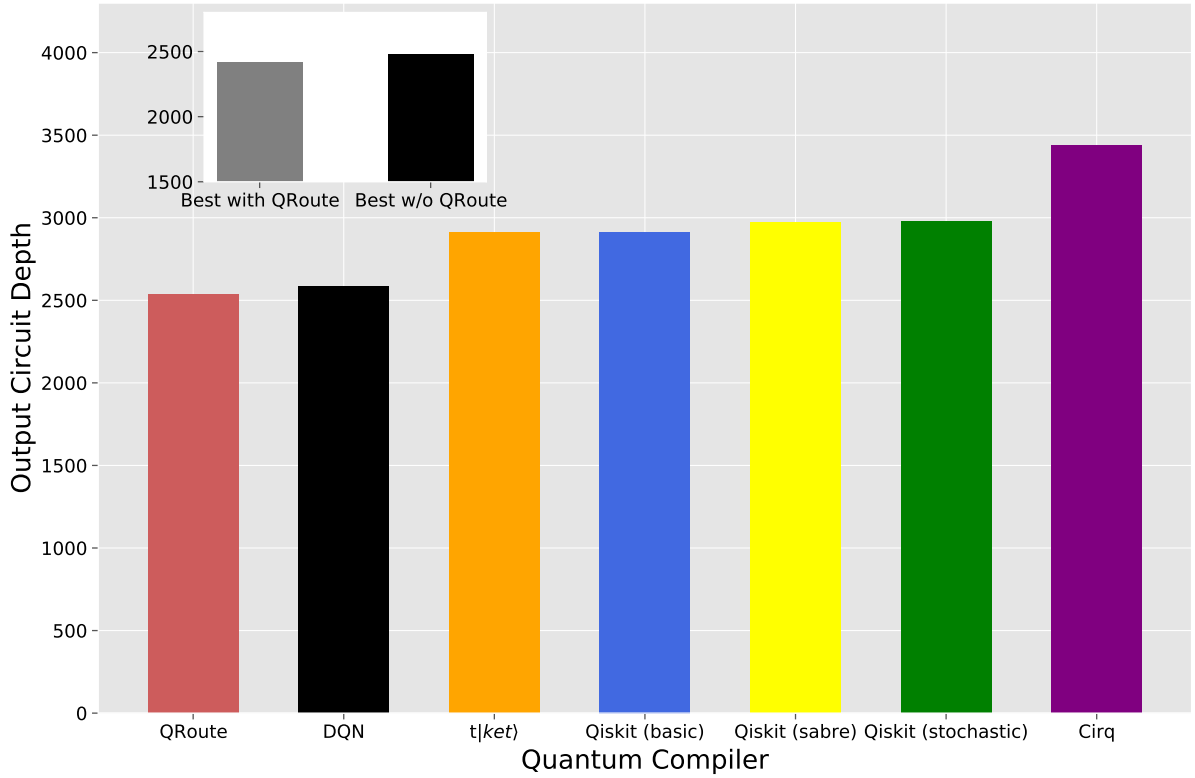


Figure 3.6: Plots of output circuit depths of routing algorithms over small realistic circuits ($\leq$ 100 gates), summed over the entire dataset. The inset shows the results on the same data comparing the best performant scheduler excluding and including QRoute on each circuit respectively.

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 [57]. 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 by [33]:

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

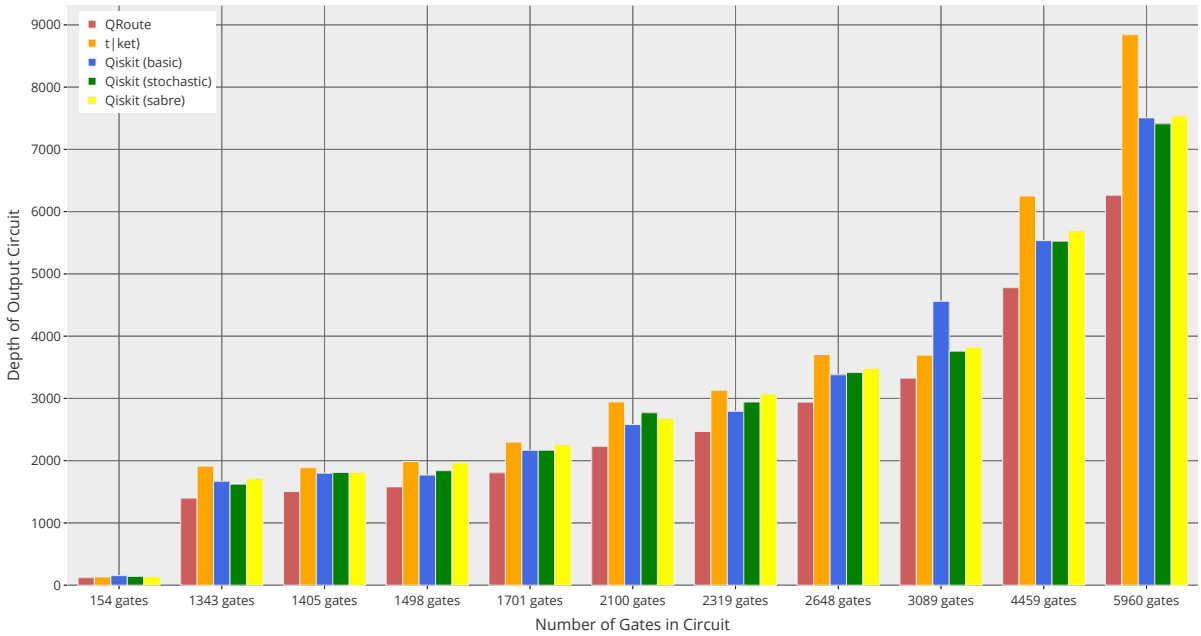


Figure 3.7: The results over eight circuits sampled from the large realistic dataset benchmark, the outputs of each routing algorithm are shown for every circuit.

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 [57]. The results are plotted in Fig. 3.7. QRoute has the best performance of all available routing methods: Qiskit and t$|$ket$\rangle$, 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 [24, 52, 53, 1, 19]. 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 [33]. 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 [16] and gate commutation rules [12] 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] [4] 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 [43]. It is integrable with other popular libraries such as Qiskit [2], Cirq [8], or PyQuil [45] and also supports instruction-set languages like OPENQasm [10] and Quil [45].

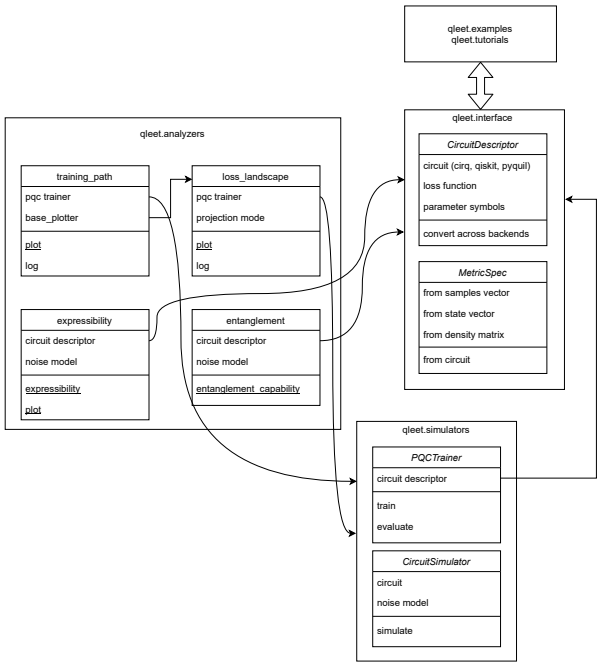
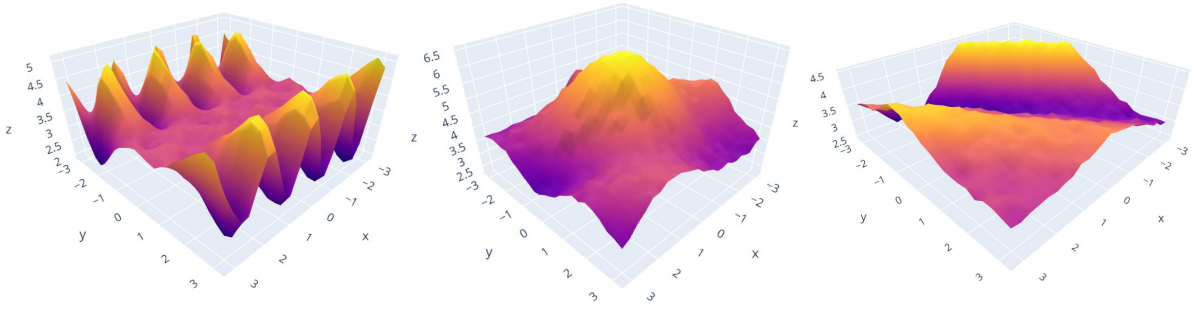


Figure 4.1: Architecture stack for qLEET

[1] https://github.com/QLemma/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:

$$\mathcal{C} = \min \text{Tr}[O\rho(\vec{\theta}_t)] \tag{4.1}$$

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

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. [20]

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.

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

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

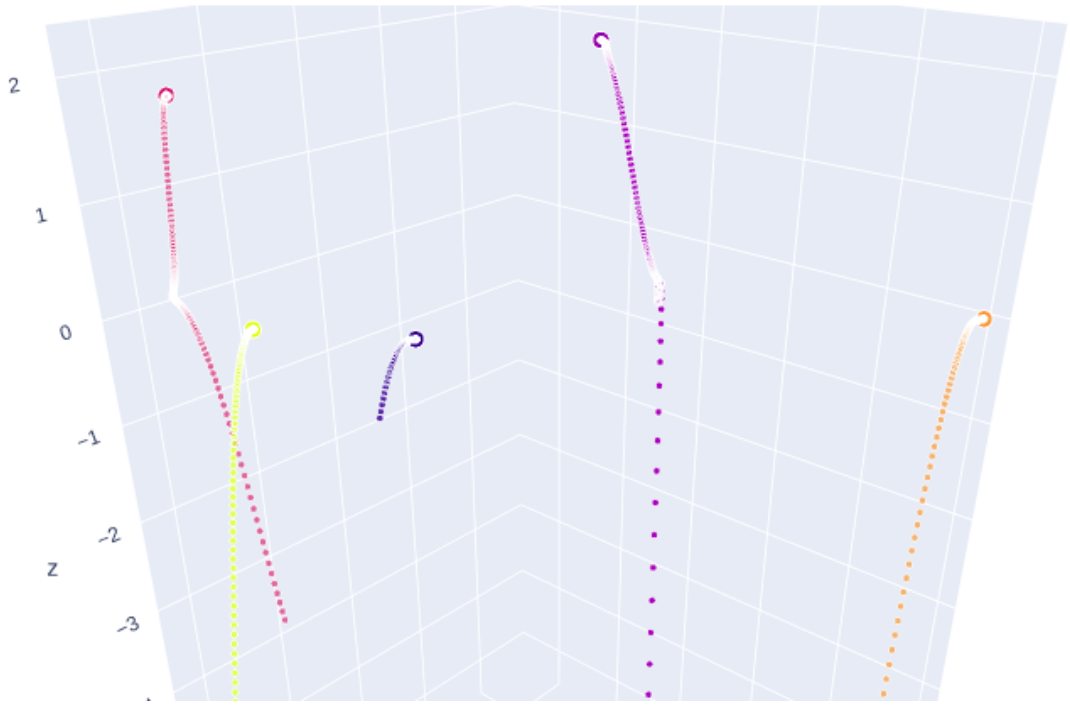


Figure 4.3: This shows a 2-D projection of the parameter vectors plotted on the X-Y axes for 5 different re-initializations, each shown using a different color. These are collected over the entire optimization run and the final value of each run is plotted with a larger blob, visible at the top of each 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.

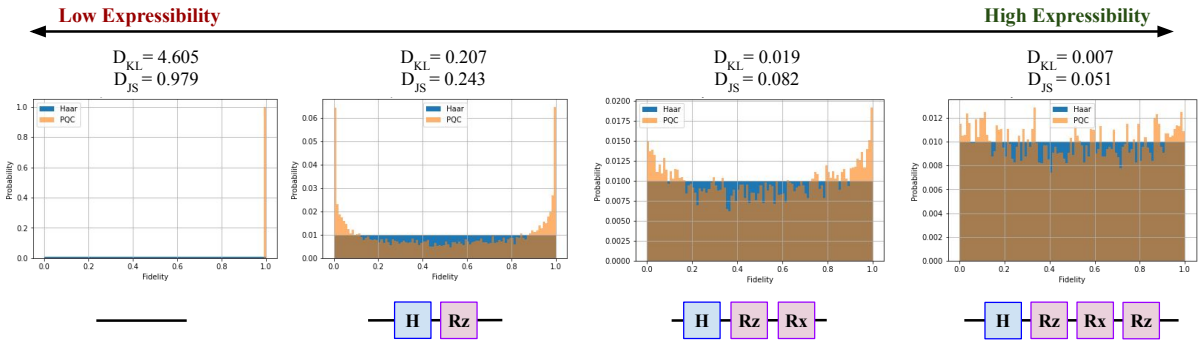


Figure 4.4: Quantifying expressibility for single-qubit circuits. For each of the four circuits show here, 1000 sample pairs of circuit parameter vectors were uniformly drawn, corresponding to 2000 parameterized states. Histograms of estimated fidelities (orange) are shown, overlaid with fidelities of 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. [22]

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.

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

where $\int_{\text{Haar}} \mathrm{d}\psi$ 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 [43], 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 [43], Meyer-Wallach Q measure [25] 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.

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

where ρ_k is the density matrix of the k-th qubit. Similarly, there is another measure called Scott Measure [23], 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:

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

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 [54, 51], which is defined as the spectrum of eigenvalues of the entanglement Hamiltonian.

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

Here, $\rho_A = \text{Tr}_B(\rho)$ 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 [56] 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 $H_{\text{ent}}^{\text{PQC}}$, 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 [49]. 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: $R_X(0, \theta_1) R_X(1, \theta_2) CZ(0, 1)$.

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

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

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.

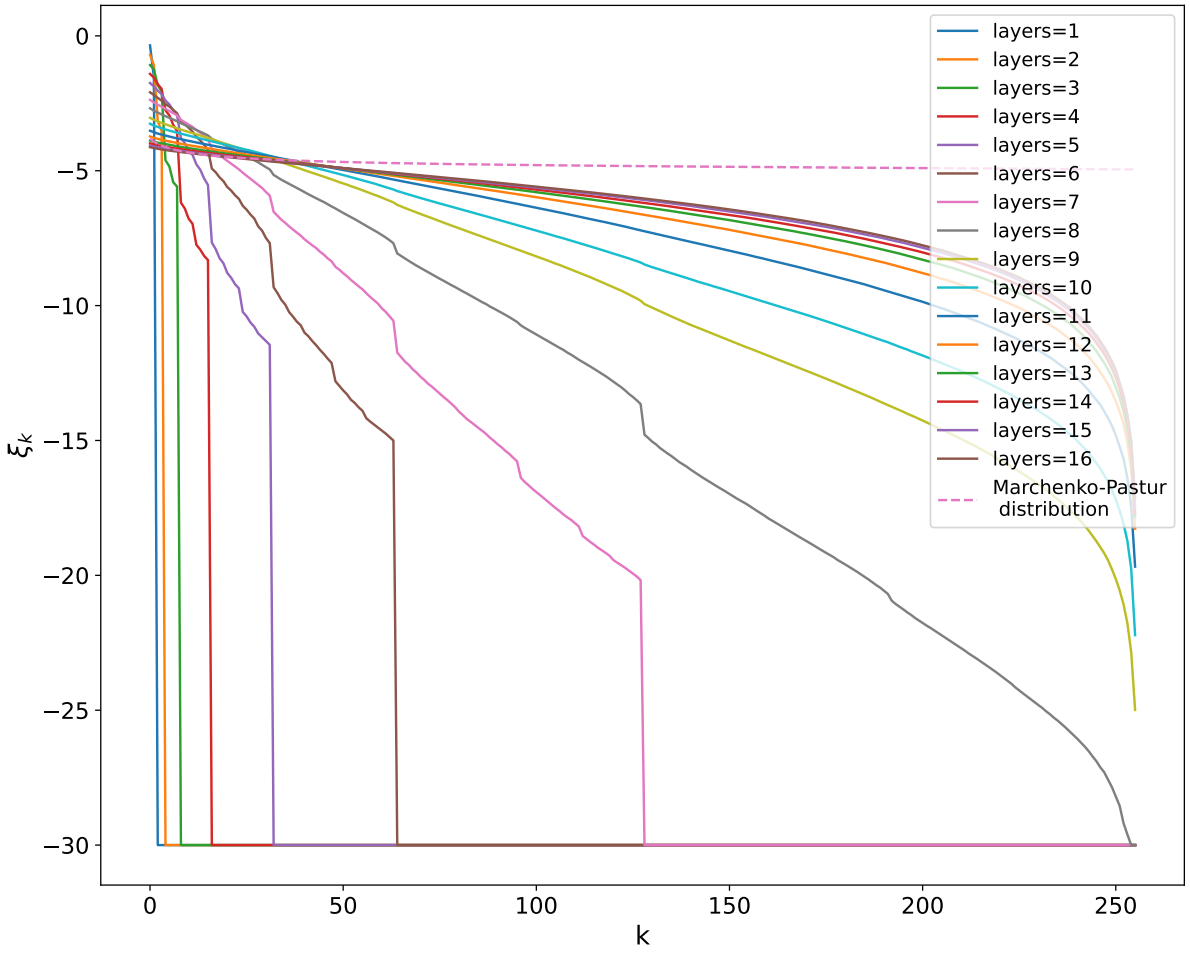


Figure 4.5: Visualizing entanglement spectrum for a PQC $U(\vec{\theta}) = \prod_l^L \left(\prod_{i=0}^{15} R_x(\theta_i^1) R_z(\theta_i^2) R_x(\theta_i^3) \dots \prod_{i=0}^{14} CX(i, i+1) \right)$. Here, ξ_k are the eigenvalues of $H_{\text{ent}}^{U(\vec{\theta})}$ arranged in descending order and 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.

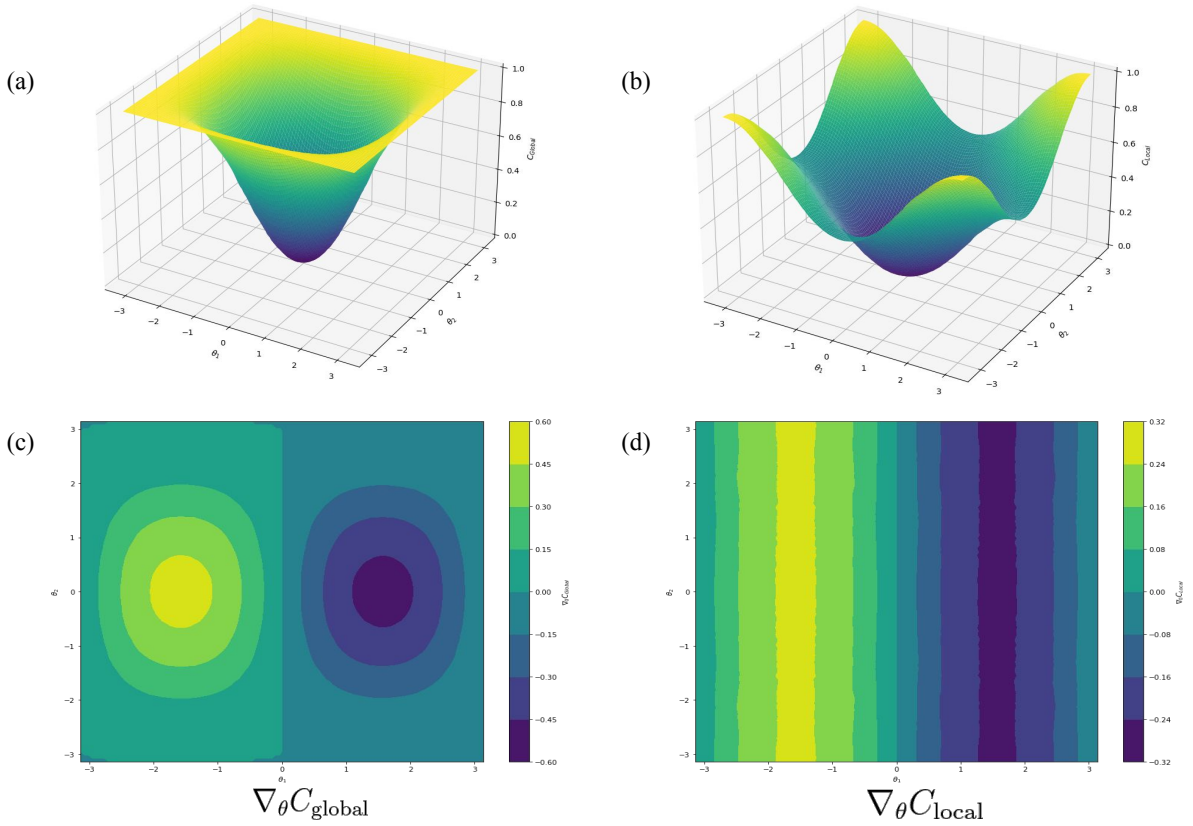


Figure 4.6: Here we show the emergence of barren plateaus in the task of learning an Identity gate using the ansatz $R_X(0, \theta_1)R_X(1, \theta_2)CZ(0, 1)$ solely based on the choice of the cost function. Figures (a) and (b) represents the loss landscape for the $\mathcal{C}_{Global}$ and local $\mathcal{C}_{Local}$ 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

Conclusion goes here ....

Related Publications

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

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.

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

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

[13] 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.

[14] 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.

[15] 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.

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

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

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

[19] 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.

[20] 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.

[21] 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.

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

[23] 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.

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

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

[26] 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.

[27] 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.

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

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

[30] 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.

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

[32] 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.

[33] 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.

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

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

[36] 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.

[37] 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.

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

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

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

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

[46] 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.

[47] 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.

[48] 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.

[49] 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.

[50] 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.

[51] 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.

[52] 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.

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

[54] 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.

[55] 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.

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

[57] 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.

[58] 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.