Quantum Circuit Optimilization Using Monte Carlo Tree Search and Reinforcement Learning

Author: Liza Darwesh
BSc. Thesis

Supervisors:
Ariana Torres, SURF
Damian Podareanu, SURF
Dick Heinhuis, HvA

Hogeschool van Amsterdam

A Preliminary Bachelor's Thesis
Graduation Assignment for SURF BV
Department of Innovation
Amsterdam University of Applied Science

June 14, 2022

Acknowledgements

Words cannot express my gratitude to my supervisors from SURF, Ariana Torres-Knoop and Damian Podareanu for the great guidance. The weekly reviews helped me to work in a more structured way and gave me the opportunity to brainstorm about problems I encountered.

I am also grateful to Bryan Cardenas Guevara, Duncan Kampert and Robert Jan Schlimbach for helping with the brainstorming and implementation of the algorithms. The hours spent explaining the algorithms by them is what I appreciate the most

I would be remiss in not mentioning my supervisor from the HvA, Dick Heinhuis for the motivational speeches and keeping my spirit high while writing this thesis.

I would like to thank Matteo Pozzi for taking the time to sit with me and demonstrate the research he has done.

Finally, I want to thank my family and my friend Gray Lyons for supporting me through this journey, believing in me, motivating me to keep going and picking out the mistakes in this thesis.

Liza Darwesh
Amsterdam, June 14, 2022

Abstract

In this thesis the question is asked: 'How can quantum circuit routing be optimized using Reinforcement Learning applied on Monte Carlo Tree Search?'. It has examined what previous studies have done to work around these problems. Looking at how the initial qubit placement procedure contributes to the optimization of quantum circuit routing has also been researched and finally, a new approach arised from the two state-of-the-art approaches was developed and has been researched on how this can help minimze the number of SWAP-gates.

For this research, in addition to researching state-of-the-art approaches, an interview has been held with one of the best-known researchers who has developed a state-of-the-art algorithm for this problem.

The results of this thesis show that circuits with minimal numbers or SWAP-gates can be found and that during the benchmark they show lower difference in depth compared to IBM's Qiskit algorithm.

In conclusion, it has become clear that the problem lies in making a quantum circuit operable on a quantum computer. The circuits must also be as small as possible, due to the decoherence of a qubit. The most successful state-of-the-art approaches to this problem are the use of the Monte Carlo Tree Search and Deep Reinforcement Learning, where the initial qubit placement is also applied. The initial qubit placement ensures that the logical qubits are mapped in the physical qubits in such a way that it is beneficial for the circuit. In that why the use of SWAP-gates is minimised. Finally, an approach has been found for minimizing SWAP-gates by using the Monte Carlo Tree Searche guided by a well trained Reinforcement Learning model.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

It is fascinating to think about the fact that the computing power of a military computer from 50 years ago, that was the size of an entire room, are in today's regular computers. However, even with the phenomenal growth we made in technology, there remain challenges which regular computers cannot seem to solve efficiently or effectively.

Quantum computers are expected to make a breakthrough in chemical and biological engineering through the discovery and manipulation of molecules; encryption for cyber security; the processing of very large quantities to aid in artificial intelligence; and the pricing of complex assets in finance [4]. Today's quantum computers perform tasks that exceed the capabilities of today's regular computers, but the world is not there yet.

1.2 Problem

Think of a spinning coin with information: either heads or tails. Eventually the coin will stop spinning and land on one of the sides. This is the same for a quantum system. A quantum computer consists of quantum bits (qubit) that have the same spinning effect as a spinning coin. In the case of a qubit, it is not heads or tails, but 0 or 1. One of the barriers to programming a quantum computer is that a qubit does not keep spinning for long. This can cause information to be lost, making some input algorithms have an unreliable output.

The algorithms that are written for a quantum computer contain a noise, which causes qubits to stop spinning faster. The way algorithms can be operated is by writing them into quantum gates. They are the same as the regular logic gates, but with quantum phenomena, which will be explained later in the paper. Mathematical algorithms are transformed into circuits of operations. These circuits are written in Quantum Assembly, a language that the quantum computer understands. Each circuit has a defined number of logical qubits. These qubits are what the gates are interacting with. In the moment of processing, the logical qubits in the circuit will then be mapped into the topology of the quantum computer, which is a connectivity architecture of the qubits inside the quantum

computer, also refereed as physical qubits. Another barrier of programming a quantum computer is matching the logical qubits with the topology, in a way that it is beneficial for the circuit. When programming a circuit, one must satisfy the topology. For instance, say two qubits need to interact with each other through a gate. If these qubits are not connected to each other, this operation will be inoperable, which means the whole circuit cannot be executed. So there needs to be a way to make circuits operable while satisfying the connectivity constraints of the quantum computer. A way to do this is by adding a so called SWAP-gate in the circuit, that causes the qubits to switch. This switch ensures that the qubits come closer to each other or even become connected. The problem with the SWAP-gate is that it is not a known logic gate for the quantum computer. This will be described later in the paper. Shortly said, there needs to be an optimization in placing these SWAP-gates to ensure that the information is reliable.

1.3 Organisation

The research of this thesis is done for the organisation SURF. A cooperative association of Dutch educational and research institutions. Universities, hospitals and research institutes work together within SURF to purchase or develop the best possible digital services. The company owns supercomputers which are available to complete high performance processes. Within the company there are multiple departments. The Innovation lab is one of these departments, which includes quantum innovation, machine learning, and high performance computing. Which will be useful for this research.

1.4 State-of-the-art

There are already a number of studies done in this area, for example a state-of-the-art research was conducted by Pozzi, et al. [5], using Reinforcement Learning and something called Quantum Annealing to optimize the number of SWAP-gates in a circuit. The research was based on a random logical qubit allocation on the physical qubits. When creating an initial qubit placement, the number of SWAP-gates can be reduced. Another recently published study was done by Sinha, et al [6], where the approach managed to minimize the number of SWAP-gates for a random qubit allocation using the Monte Carlo Tree Search in combination with a Graph Neural Network.

1.5 Research

Both papers discussed the need to look into initial qubit placement, which will also ensure that the SWAP gates are minimized in addition to using Reinforcement Learning. With this in mind, research will be conducted on quantum circuit optimization using the Monte Carlo Tree Search in combination with Reinforcement Learning. An initial qubit placement will be used for this.

This paper answers the main question: 'How can quantum circuit routing be optimized using Reinforcement Learning applied on Monte Carlo Tree Search?' This question will be answered by means of the following sub-questions:

- Why are there problems with quantum circuit routing?
- What has already been done to successfully accomplish quantum circuit routing?
- How will an initial qubit placement procedure help with the minimization of SWAP-gates?
- How does the Monte Carlo Tree Search work on optimizing the number of SWAP-gates?

This research paper is organised as follows. Section 2 offers a description of basic knowledge about quantum computing phenomenon and techniques needed for understanding the main question. Section 3 demonstrates what research has already been done by others. Section 4 describes the sub-problem initial qubit placement and how to solve it. Section 5 provides an overview of the experimental setup, which is how the Monte Carlo Tree Search is put together with the Reinforcement Learning. Section 6 shows the results of what the reinforcement learning model delivered. Finally, in Section 7 and 8 states the discussion about the research that is performed and the final conclusion.

Chapter 2

Quantum Circuit Routing

It is recognized that quantum computers can provide revolutionary developments. This chapter gives answer to the question 'what are the problems with quantum circuit routing' by first explaining what a quantum computer is and how it can be programmed. After that, an explanation will be given on the problem with an understanding of what needs to be optimized.

2.1 The Qubit

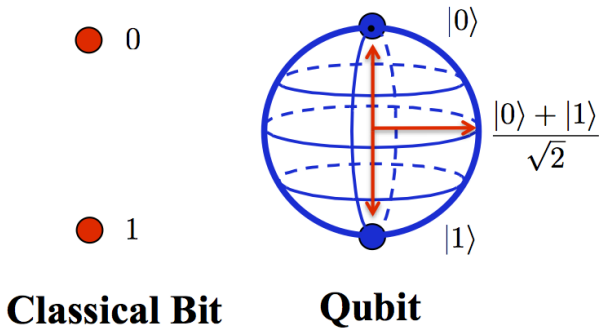

Figure 2.1: Regular bit and quantum bit illustration

A quantum computer can be seen as a regular computer that stores information and performs operations using quantum mechanics. A regular computer stores all its information such as numbers, text, and images, in series of 0's and 1's. These units are called bits. The way a quantum computer stores information is through the use of quantum bits (qubits). The difference between a bit and a qubit is that a qubit can carry not only the information 0 or 1, but also 0 and 1 on top of each other. This phenomenon is called superposition. A well-known example of superposition is the Schrödinger's cat example, where a hypothetical cat is illustrated in a closed box with a bottle of poison. It is not certain whether the cat is alive or dead unless it is checked. Before checking, the cat finds itself in a situation where it has a 50% chance of being dead or alive. This is the same with a qubit, there is no certainty whether it will hold 0 or 1, only a 50% probability that it will be one of the two. See Figure 2.1 for an illustration of a bit and a qubit.

2.2 Quantum Circuits

A regular computer operates by means of directions (algorithm) that are provided in the form of a script. This script then tells the computer processor what calculations have to be done to arrive at a certain output. These scripts can be written in different programming languages and the computer translates this into binary language, to control the logic gates inside the processor. The logic gates provide the calculations necessary for the operations. This principle, of programming gates that do calculations to perform a certain operation, work the same in a quantum computer. A script is written that determines which quantum gates should operate. An example of a gate that will be discussed frequently in this thesis is the Controlled NOT gate (CNOT-gate), see figure 2.2.

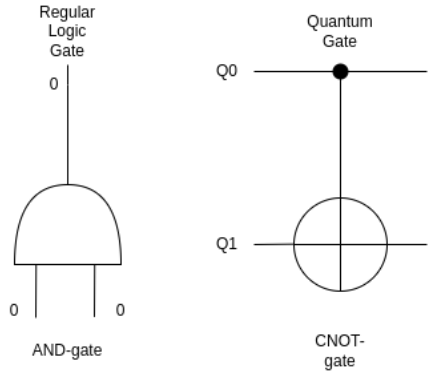

Figure 2.2: Example of a logic gate in a regular computer (left), example of a Quantum gate (right)

This script then creates a circuit consisting of the sequence of the quantum gates that were defined in the script. The algorithm that has been programmed is thus represented as a circuit with quantum gates. An example of a circuit can be seen in figure 2.3. The circuit illustrates between which qubits operations have to be performed. The circuit of figure 2.3 illustrates a four qubit circuit consisting of CNOT-gates with each interacting with two qubits. Usually, qubits are represented in Dirac notation [7], but in this thesis that is not relevant and the qubits will be represented with a capital Q.

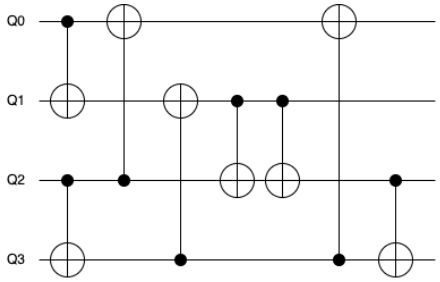

Figure 2.3: A quantum circuit extracted from a programmed script

Each horizontal line in the circuit indicates the 'path' of a qubit, each path can have a quantum gate through which the qubit has to pass. Ultimately, the paths can be read to obtain a final result. As mentioned earlier in the example

of Schrödinger's cat, there is a 50% chance that the cat is alive or dead unless the box is opened. Reading or measuring a circuit forces the qubits to choose a state, either 0 or 1. The moment it chooses one of the two, the qubit collapses to a regular bit with the one value. This means that the output of the circuit will represent an ordinary series of bits, just like in a regular computer.

2.3 Topology

The step after feeding a quantum circuit to the quantum computer is the process in which the defined qubits in the circuit (logical qubits) are allocated to the qubits in the hardware (physical qubits). The logical qubits are, as it were, assigned a location on the hardware. The form in which these qubits can be allocated depends on the connectivity architecture of the physical qubits. This means that the qubits in the hardware are attached to each other in a certain form. This architecture is also known as the topology of the quantum computer.

There are different types of topologies. For example, figure 2.4b shows the topology of Rigetti's Aspen-4 quantum computer with a linear structure, or figure 2.4e shows the topology of Google's Sycamore, which consists of more connected qubits in a grid structure.

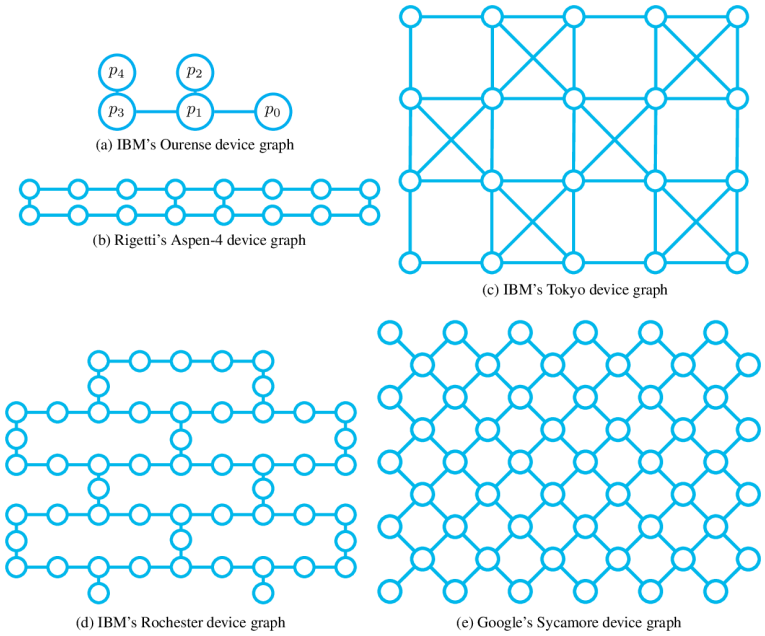

Figure 2.4: Quantum topologies of big tech companies' quantum computers. Source: [1]

These connectivities determine which qubits are allowed to interact with each other. Imagine the topology of figure 2.5 and the circuit of figure 2.3.

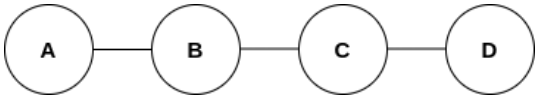

Figure 2.5: Example of a linear topology.

There are several ways to map the circuit's logical qubits into the computer's physical qubits, see figure 2.6. It can be done in sequence, randomly, but also with an initial placement, where the allocation is based on the qubits that need to interact in the circuit.

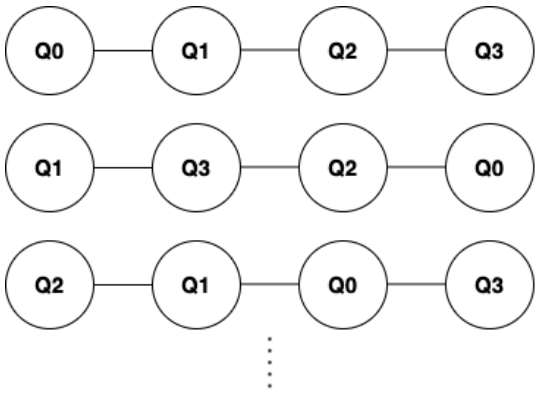

Figure 2.6: Different combinations of qubit mapping.

After mapping the logical qubits, the operations in the circuit will be performed. The moment interactions have to be performed in the circuit between qubits that have no connection in the topology, the operation cannot be performed, after which the entire circuit is rejected by the quantum computer. This appears to be a common problem when writing quantum algorithms. One way to make the circuit perform on a quantum computer is to add SWAP-gates to the circuit 2.7.

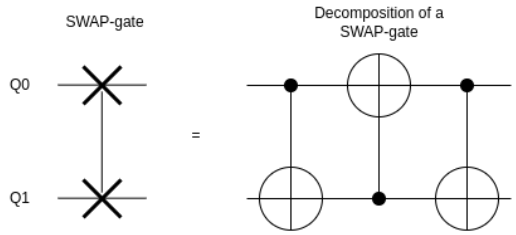

Figure 2.7: A SWAP-gate and its decomposition.

The SWAP gate will ensure that the qubits, that interact with the SWAP-gate, will switch positions in the quantum computer. This means that the qubits that have to operate with each other have a connection in the topology. This can be done over the entire circuit, until all operations are executable hardware-based.

A SWAP-gate is not a standard gate known to the quantum computer. A SWAP-gate can be simulated by placing three CNOT gates in a row, as shown in figure 2.7. Because there are actually three extra gates added per SWAP gate, this will ensure that the circuit will have a greater depth.

2.4 Decoherence

Qubits can be compared to a spinning coin. Eventually the coin will stop spinning and land on heads or tails. A qubit is also spinning between 0 and 1, but will also

eventually stop spinning, after which it will end up at 0 or 1. This phenomenon of breaking out of the superposition, after which it has to choose a state is called decoherence. The lifespan of a qubit differs per hardware composition of the quantum computer. Based on the temperature and network in the quantum chip, it can be ensured that the lifespan of a qubit is longer. The longest achieved so far by a qubit is 50 seconds, but the most common is somewhere between 15-120µ seconds [8]. Because qubits decohere, it causes parts of information to be lost. As a result, the output information will be unreliable. For this reason, it is important to set up a circuit that is small in depth, so that it can be quickly executed by the quantum computer while the qubits last. This is a tricky problem when a large algorithm has to be executed, where SWAP-gates are essential.

Because it is important to have a circuit that can be executed on a quantum computer, but also has to be small in depth, it is necessary to look for a way to optimize quantum circuits. A number of approaches have already been undertaken for this purpose. These can be found in the next chapter.

Chapter 3

Previously Researched Work

This chapter will provide a picture of what has been done before to optimize the quantum circuit routing and what has been recommended by these researchers for further research.

3.1 Cambridge's $t\,|ket\rangle$'s Routing Algorithm

Several approaches have been taken to optimize the circuit routing problem. They have all shown a state-of-the-art application. A well-known one is the use of the routing algorithm from Cambridge's Quantum Software Development Kit (Q-SDK) $t\,|ket\rangle$ [9]. The $t\,|ket\rangle$ Q-SDK is designed to maximize the performance of quantum algorithms when run on quantum computing hardware and accelerate the development of quantum computing applications across multiple industrial sectors. Cambridge students construed a solution to the circuit routing problem implemented with $t\,|ket\rangle$ [10]. They believe that this heuristic method in $t\,|ket\rangle$ matches or outperforms the results of other circuit mapping systems in terms of depth and total gates of the compiled circuit, and has a much shorter run time allowing larger circuits to be routed. They envision a quantum computer as a graph in which nodes are qubits and edges are the allowed interactions between the qubits. Their new architecture-agnostic methodology for mapping quantum circuits to real-world quantum computers with limited qubit connectivity present empirical results demonstrating the effectiveness of this method.

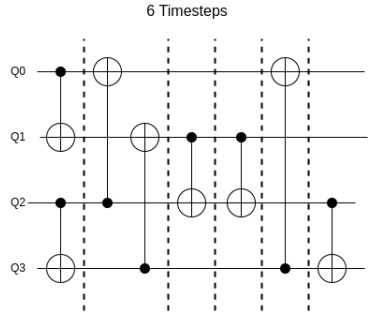

Figure 3.1: Example of a quantum circuit containing two-qubit gates acting on four qubits, Q0, Q1, Q2 and Q3. This circuit has six timesteps, each with gates acting on disjoint sets of qubits.

A procedure was defined by $t\,|ket\rangle$ for the initial mapping of logical qubits into

physical qubits. They repeat all the gate operations to construct a graph whose vertices are qubits. The gate operations in a circuit can be separated by means of boundaries, see Figure 3.1. Each separation is referred to as a timestep. At each time step, they add an edge to the graph between the interacting qubits present in the timestep. Decoupled qubits in the graph correspond either to qubits that have no interaction at all, or to qubits whose first interacts with a qubit whose first two interacts with others. These decoupled qubits are not included in the initial placement. They are added later in the routing procedure. This graph will eventually have nodes with multiple edges. The edge with the most edges is allocated first. The way this is done, will be explained in chapter 4. This ensures that most of the gates in the first two timesteps can be applied without any SWAP-gates. If the initial mapping cannot be completed respectfully to the topology, consideration should be given to the placement of SWAP-gates.

3.2 SWAP-based BidiREctional heuristic search algorithm

The next approach is inspired by the paper written by Gushu Li et al. [3] where they describe, a SWAP-based BidiREctional heuristic search algorithm, named SABRE, proposed to solve the qubit mapping problem. IBM's Q-SDK, called Qiskit, created a function part of a transpiler library based on the algorithms described in their paper. Transpilation is the process of rewriting a particular input circuit to match the topology of a particular quantum device, and/or to optimize the circuit for execution on today's noisy quantum systems. Most circuits must undergo a series of transformations that make them compatible with a particular device and optimize them to reduce the effects of noise on the resulting results. Rewriting quantum circuits to meet hardware limitations and optimizing for performance can be far from trivial. The flow of logic in the rewrite tool chain does not have to be linear and can often have iterative subloops, conditional branches, and other complex behaviour. That being said, the basic building blocks follow the structure given in Figure 3.2.

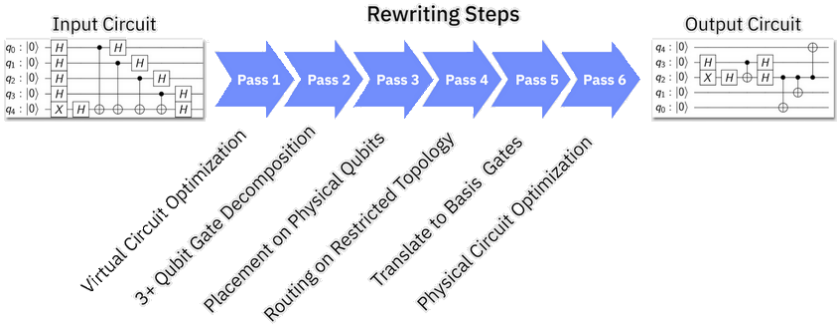

Figure 3.2: IBM's Transpilation Processes for Circuit Optimization, Source: [2]

The algorithm aims to minimize the number of SWAPs inserted and the circuit depth. This algorithm starts with an initial layout of logical qubits on physical qubits and repeats the circuit until all gates are exhausted, inserting SWAPs along

the way. This algorithm starts with Pass 3 from the process shown in Figure 3.2 and goes to Pass 1 after the initialization of the qubit placement. In each iteration, it is first checked whether there are gates in the front time step that can be applied directly. If so, it will apply them and check for the next timestep. Otherwise, it will try to search for SWAP-gates, insert the SWAP-gates, and update the mapping. The search for SWAP-gates is limited in that Qiskit considers only physical qubits close to those qubits in the previous time step.

Observing that many exhaustive search attempts may be redundant and that effective initial placement must begin with the qubits in the gates to be executed, Li et al. designed an optimized SWAP-based heuristic search scheme in SABRE with significantly reduced search space. They present a novel search technique in SABRE to naturally generate a initial mapping through traversing a reverse circuit, in which more consideration is given to those gates at the beginning of the circuit without completely ignoring the rest of the circuit, see Figure 3.3 .

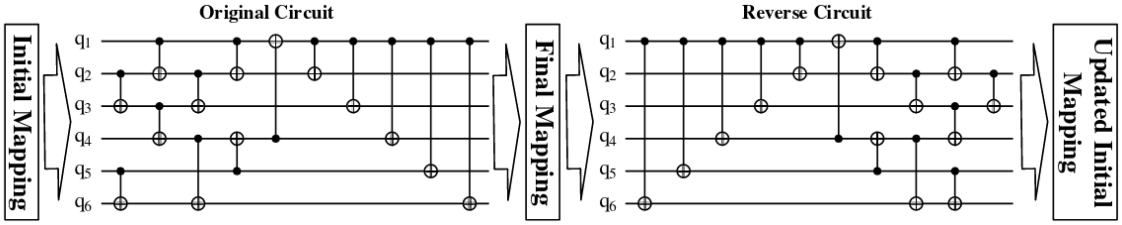

Figure 3.3: Initial Mapping Update Using Reverse Traversal Technique, Source: [3]

In addition, they introduced a heuristic cost function for evaluating overlapping SWAP-gates, so SABRE tended to select non-overlapping SWAP ports. This optimization generates several hardware compatible circuits with a trade-off between circuit depth and number of gates.

3.3 Reinforcement Learning Approaches

The most popular approach for circuit routing optimization is through deep learning with reinforcement. In the article written by Matteo Pozzi et al. [5], they describe a qubit routing procedure using a modified version of the deep Q-learning model. Where they benchmark with the most advanced quantum compilers currently available (Qiskit and t|ket), on both random and realistic circuits. This research was a follow-up to the original research by Herbert and Sengupta [11]. Introducing the concept of reinforcement learning (RL) for the qubit routing problem. The way the RL model learns is through feeding the agent (neural network model) the current state and the reward that comes along with it. For this research, the researcher Matteo Pozzi has been interview through a zoom call to get an understanding of how the algorithm works. Through a live demo the following has been explained. That way can the model can decide what action to take next, see Figure 3.4. The concept of RL will be described in more details in chapter 5.

The state that is represented to the agent, defined by the writers, contains the following information: the initial location of the qubit, its interaction with

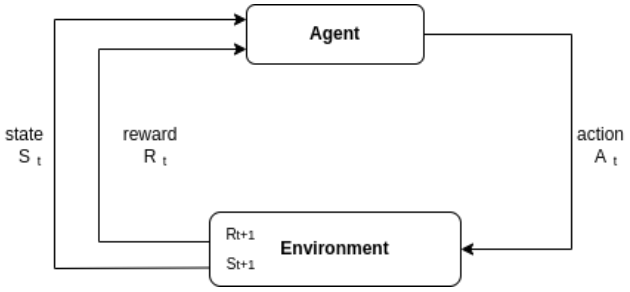

Figure 3.4: An illustration of the RL process

another logical qubit, and if it is already possible to carry out a CNOT-gate immediately. The agent has to select some SWAP-gates of the remaining qubits that are not involved in the CNOT-gates. By training a model, they try to carry out the routing. The state representation goes through a function that computes a feature representation, which condenses the state in a fixed length vector of distances (how many qubits does the model have to go through). They feed the vector through the neural network, which outputs a single continues number which is the Q-value (quality of the current state) in the context of Q-Learning, a learning reinforcement learning technique, based on the quality of state and next state. They use the concept of simulated annealing to select a bunch of SWAPs to add the action in sequences on possible SWAP-gates and evaluates the quality. This process is depicted in Figure 3.5. Simulated Annealing is a simulation of the thermodynamics Annealing concept. Simply explained, annealing is where the system ends up in better solution, in a landscape of solutions, through a fast decrease in temperature of the system. The simulated annealing encourages the exploration of the state space and decreases the temperature, so the system ends up exploring less and less, until it ends up only adding a SWAP-gate when it is benefiting the quality.

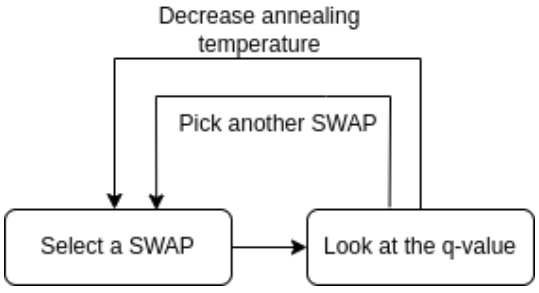

Figure 3.5: Q-Learning process with Simulated Annealing

They mentioned that in order to achieve this approach, it is necessary to alter the conventional RL framework to allow combinatorial action space, where they mention numerical results that are included to demonstrate the advantage of the RL-trained qubit routing policy over using a sorting network. Their approach turned out to be one of the best approaches, with results that turned out to be better than the approaches used by IBM and Cambridge.

3.4 Graph Neural Network with Monte Carlo Tree Search

The last approach is the use of the Monte Carlo Tree Search (MCTS) guided by a Graph neural network. In the paper written by Sinha et al. [6] they mention that the depth of the transformed quantum circuits is minimized by utilizing the 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 and action probabilities for each state. The way MCTS is briefly shown in Figure 3.6. The way MCTS works is more thoroughly explained in chapter 5.

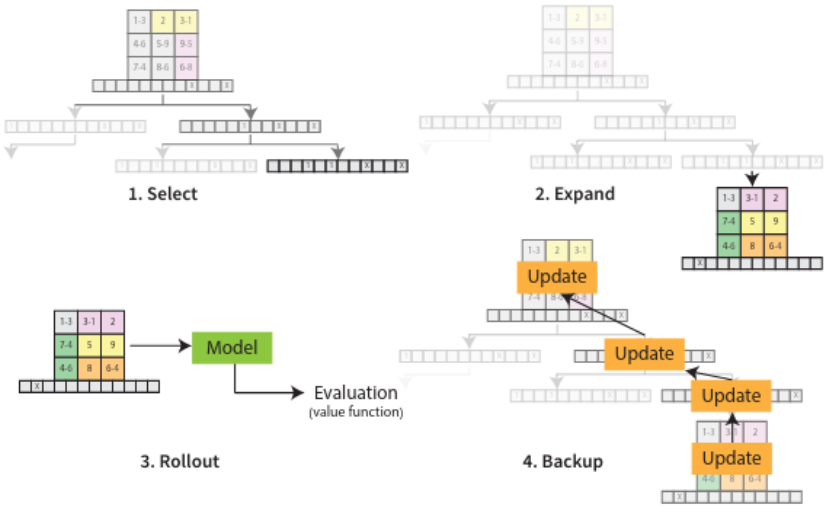

Figure 3.6: Iteration of a Monte Carlo tree search: (1) select - recursively choosing a node in the search tree for exploration using a selection criteria, (2) expand - expanding the tree using the available actions if an unexplored leaf node is selected, (3) rollout - estimating long term reward using a neural network for the action-state pair for the new node added to search tree, and (4) backup - propagating reward value from the leaf nodes upwards to update the evaluations of its ancestors.

Along with this, they propose a new method of adding extra variables in their state representation which helps factor in the parallelization of the scheduled operations, thereby pruning the depth of the output circuit. The way parallel execution works is shown in Figure 3.1, where it is possible to execute multiple gates in one timestep, because the qubits they interact with are not already in a different operation. They referred to their procedure as QRoute, where it performs qubit routing in a hardware agnostic manner, and it outperforms other available qubit routing implementations on various circuit benchmarks.

Yet, they mention that an optimization in runtime can be done through the use of a different kind of neural network approach. QRoute also shares an issue in mitigating 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 CNOT-gates directly into their neural network by using transformer encoders to handle the arbitrary length sequence data. 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, they note that QRoute is a primitive implementation of their ideas, and there is great scope of improvement in future.
Many different state-of-the-art approaches have been undertaken by researchers. The Monte Carlo Tree Search approach is one of the newer approaches that, in combination with Reinforcement Learning and an initial qubit placement, can lead to remarkable results. This is something that needs to be explored. In the next chapter, the initial qubit placement will be further explained.Many different state-of-the-art approaches have been undertaken by researchers. The Monte Carlo Tree Search approach is one of the newer approaches that, in combination with Reinforcement Learning and an initial qubit placement, can lead to remarkable results. This is something that needs to be explored. The initial qubit placement will be explained in more detail in the next chapter.

Chapter 4

Initial Placement of the Qubits

This section will give answer to the question 'How will an initial qubit placement procedure help with the minimization of SWAP-gates?' by introducing the qubit allocation problem and giving a understanding of some of the possible approaches to solve this problem.

4.1 Qubit Allocation

Given a quantum circuit and an architecture, there must be a way to see if it is possible to map the logical qubits in the former to the physical qubits in the latter. This is called initial qubit placement. The moment that an initial qubit placement ensures that all operations in a circuit are possible, the use of SWAP-gates is no longer necessary. However, it is possible that not all gates can be executed. The moment a mapping is set up in such a way that, for example, 80% of the gates are operable, it may be that fewer SWAP-gates are needed to apply, than when a random allocation is performed. In the paper written by Siraichi et al. [12] a mathematical definition is given for the qubit allocation problem. A simplified explanation would be, suppose there is a topology of the coupling between the physical qubits, of which a list ϕ is made with qubits that interact with each other and a list ψ with qubits involved in a CNOT-gate operations of the circuit. The moment an order of ψ can be arranged in such a way that most or all CNOT-gate operations become possible. The solution given by them has been applied to a Directed Acyclic Graph Topology. Which means the direction of the edges matters. To illustrate this, assume the topology of Figure 4.1.

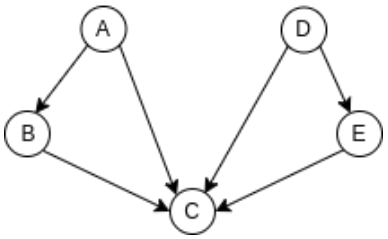

Figure 4.1: Toy topology for the initial qubit placement

And circuit interacting on this topology illustrated in Figure 4.2.

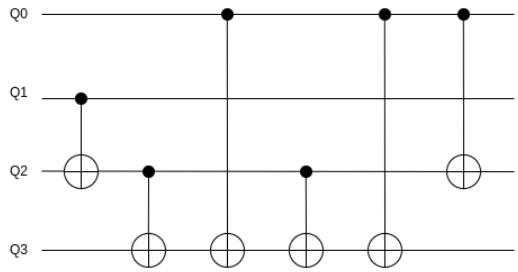

Figure 4.2: Toy circuit model of CNOT gates for the initial qubit placement

4.2 Weighted Directed Acyclic Graph

To devise a way to arrange the logical qubits in the topology, in such a way that all CNOT-gate operations are possible, it is necessary to look at what kind of topology is expected. This is done by creating a directed acyclic graph based on, based on the out-degree of the qubits. An out-degree is the number of edges going out from a node. So it is counted how many interactions each qubit has to engage in. Using the example of circuit 4.2, the Directed Acyclic Graph 4.3 can be made, illustrating the operations that need to be performed.

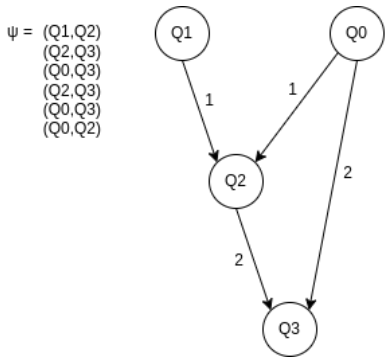

Figure 4.3: Coupling Directed Acyclic Graph of the toy circuit model

In the list ψ all CNOT-gate operations are written out in a set with the interacting qubits. What can be inferred from this graph is that when all the out-degrees of every qubit in the graph (the number of arrows) are added up, a logical order of most out-degree would be $Q0 - Q2 - Q1 - Q3$. Another thing that can be made out is the different interactions that each qubit has. For example, it is clear that Q0 undergoes two different interactions, with Q2 and Q3. From this we can already say that Q0 must come to a position where the physical qubit interacts with two other physical qubits. It can also be established that Q3 itself has no out-degrees, so Q3 must be allocated in a position in where there are no out-degrees either.

If the out-degree criterion is used as stimulus to map the logical qubits to physical qubits, it becomes possible to make the CNOT-gate operations executable. The table below shows an overview of the number of out-degrees of each node in the topology.

Node	number of out-degrees
A	2
B	1
C	0
D	2
E	1

Qubits	number of out-degrees
Q0	3/2 (3 in total, 2 unique)
Q1	1
Q2	2/1 (2 in total, 1 unique)
Q3	0

When the topology of the physical qubits and the Weighted Directed Acyclic Graph are combined, an allocation like the one shown in Figure 4.4 would yield based on the matching of the number of out-degrees.

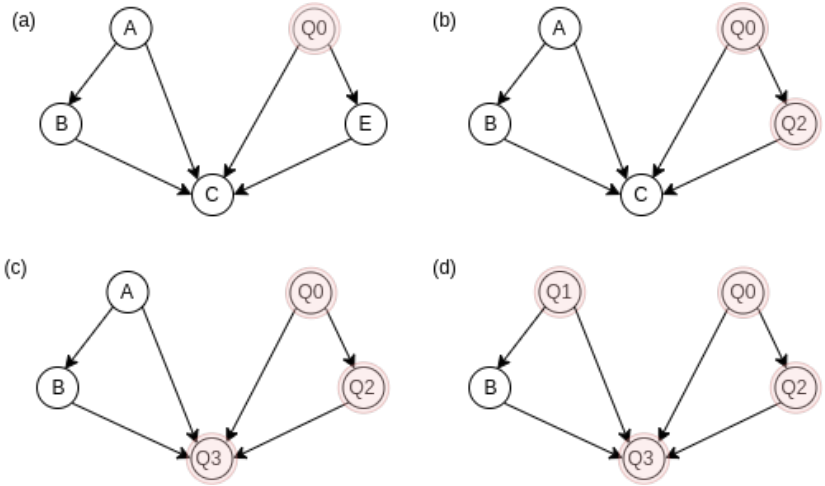

Figure 4.4: Initial qubit placement approach on a DAG by using the highest out-degree value and matching that with the coupling graph of the circuit.

Because Q0 has the most out-degrees, it is allocated first. Since it only has two unique out-degrees, it is matched with a node that also has two out-degrees. In this case, that would be node A and D. Because the topology maintains a mirror image, a node is randomly chosen to allocate Q0 on. In the case of Figure 4.4(a), it is allocated to node D. The next qubit with the highest out-degree is Q2, which according to the Weighted Directed Acyclic Graph in Figure 4.3 has an edge with Q0. A logical location would be node E (Figure 4.4(b)), because it is connected to Q0 and despite having 2 total out-degrees, it has only 1 unique

out-degree. The qubit allocated next is Q3. The reason that Q1 does not act first is, because Q1 has a directed edge to Q2, but is not reachable. Q3 would be a simple allocation, as it matches the location of node C (Figure 4.4(c)). They both have an out-degree of 0 and do not point anywhere. There are two qubits pointing to Q3, so it makes sense to allocate it in the middle node. Next, the shortest route to Q2 is considered for the allocation of Q1. In this case, the route from node A and B is the same, so the allocation will have to be chosen arbitrarily. In the case of Figure 4.4(d) node A has been chosen. Since Q1 has no direct edge to Q2, this operation is inoperable. In this case, the next step is to use SWAP gates to have the two qubits contiguous, making the circuit executable.

4.3 Weighted Undirected Graph

In the case of an Undirected Graph topology, the direction of each edge no longer matters. In this case, the tie-breaking decision can be made based on the number of edges, independent of the direction. Assume the Undirected Graph topology from Figure 4.5.

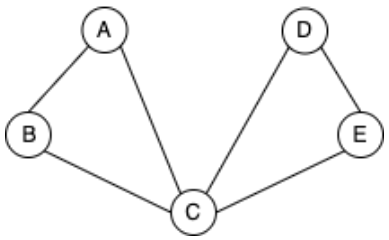

Figure 4.5: Undirected Graph Topology

Again, each node has its own number of edges. The table below gives a representation of this.

Node	number of out-degrees
A	2
B	2
C	4
D	2
E	2

As shown in the table, there are many nodes that have the same value of edges. This makes the initial placement procedure difficult. This small example is a simple representation of the real problem. The moment logical qubits have to be mapped into a topology, such as Google's Sycamore device graph (Figure 2.4), things get a lot more difficult. Figure 4.6 shows the Nondeterministic Polynomial time complexity classes. In Computer Science the performance or complexity of an algorithm is described in certain time complexity. The time complexity describes the worst-case scenario, and can be used to describe the execution time required by an algorithm. Easy algorithms run in constant time, but large algorithms for example can quickly be expensive in time. The moment an algorithm

has a polynomial time complexity, this means that the algorithm is heavy in computing power for the computer. Polynomial complexity is a sub category of the Nondeterministic Polynomial (NP) time complexity class, which can be seen in Figure 4.6. The higher up in the intersections, the harder it is for the computer to execute the algorithm. NP-problems can usually still be performed by a computer, but the moment a problem is NP-Complete or NP-Hard, enormously heavy computing power is required from the computer to execute the algorithm.

The problem of initially mapping qubits in a large topology is known as an NP-Complete problem. There appears to be no straight-forward solution for the initial placement procedure problem on large topologies, so that randomness is often still needed to help determine the mapping of the logical qubits.

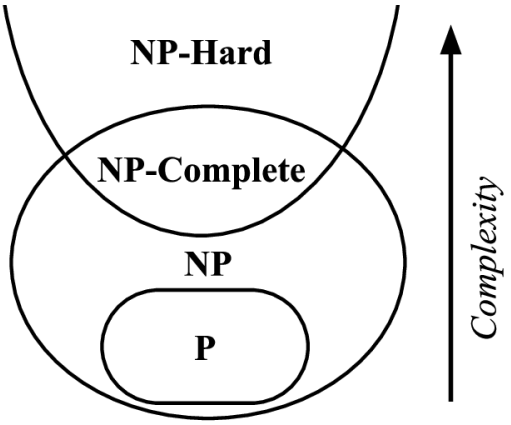

Figure 4.6: Nondeterministic Polynomial time complexity classes

In Figure 4.7 the same sketch is made of a way in which the qubits can be mapped on the basis of matching the number of edges. In this case it is possible to make the interaction between Q1 and Q2, because the problem of the directions of the edges no longer matters. This ensures that all operations in list ψ (Figure 4.3 can be performed, which means that the circuit in Figure 4.2 is operable without using any SWAP-gates.

4.4 Subgraph Isomorphism Partitioning

The process of checking whether two graphs fit together is called Isomorphism in mathematics. Because the topology is often larger than the weighted graph, that arises from a circuit, the problem that is faced is called Subgraph Isomorphism. It is checked whether a part of a large graph matches another smaller graph, see Figure 4.8.

Initial qubit placement checks whether the pattern of the subgraph fits into the pattern of the topology graph. The qubit allocation problem can be broken down into two subproblems: (a) Subgraph Isomorphism and (b) Token Swapping problem. Token Swapping is all about finding the shortest route to a target, in this case finding the minimum number of SWAP gates leading to a working circuit.

In a topology where most edges have the same outdegrees, this can cause the solution space to be too large. In most cases a complete weighted graph, like in

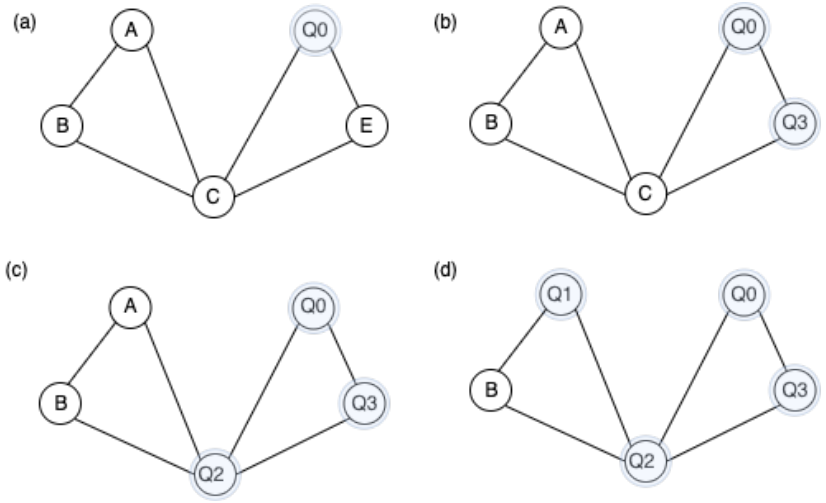

Figure 4.7: Initial qubit placement with a combination of in-degrees and out-degrees of the vertices and matching those with the coupling graph of the circuit.

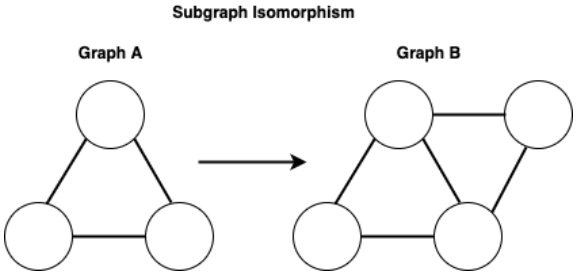

Figure 4.8: Illustration of a Subgraph Isomorphism

Figure 4.3, it is unlikely for the graph to be Isomorphic with its topology graph. Sirarchi came up with an update to his approach [13]. A this point an option for this problem can be to look at Operation Partitioning, where CNOT operations in the list are divided into pieces and Subgraphs are made of these operations. By checking these Subgraphs on Isomorphism, an overview is created of the qubits, which are related to each other, that can be mapped directly in the topology. Figure 4.9 shows how this works.

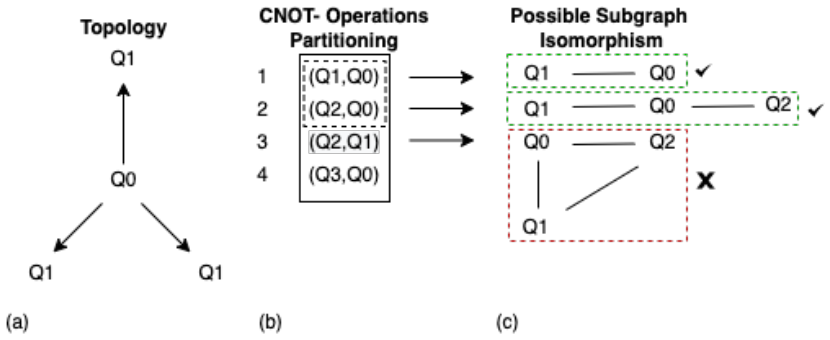

Figure 4.9: Example of a Subgraph Isomorphism Partitioning

Take the topology in Figure 4.9(a) and the list of CNOT operations (b). By making Subgraphs of the operations, the Isomorphism can be looked at. The moment a Subgraph is not Isomorphic with the topology, another mapping has to be looked at and the maximum number of qubits that are Isomorphic with the topology is remembered. From this point, a mapping can be created from the Subgraph with the most qubits. From this point, it is possible that not all operations satisfied. At this point, the tie-breaker will be the Token Swapping, which will complete the mapping by adding SWAP-gates to make the other CNOT-gates possible. The way the Token Swapping is handled is through the use of the Monte Carlo Tree Search, which is further explained in Section 5.

With these approaches it is possible to ensure that number of SWAP gates are minimized. By devising an allocation that is beneficial for the circuit, it can be ensured that the addition of SWAP-gates does not have to be necessary. The next chapter explains how to add a SWAP gate using the Monte Carlo Tree Search in combination with Reinforcement Learning when it is necessary to add a SWAP-gate.

Chapter 5

Monte Carlo Tree Search with Reinforcement Learning

This chapter will describe what the Monte Carlo Tree Search (MCTS) and Reinforcement Learning is and how a combination of the two could lead to an optimization in the number of SWAP-gates added to a quantum circuit.

5.1 Implementation of the MCTS

In 2016, the computer program AlphaGo was nominated as the breakthrough of the year. The program is a combination of advanced MCTS with deep neural networks and is made for the Asian game Go [14]. The idea of using the MCTS for the routing of the quantum circuits is inspired by AlphaGo, where all possible actions are simulated, until a best answer is obtained. MCTS is used to determine follow-up actions. To give an illustration of how it works, imagine the game Tic-Tac-Toe, where the best actions for crosses must be be decided 5.1.

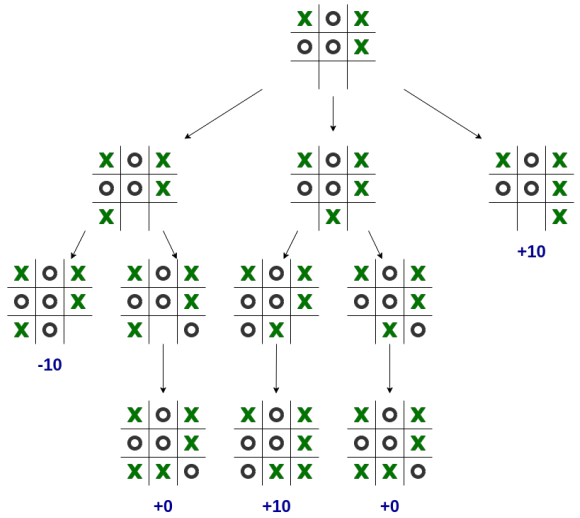

Figure 5.1: Simple illustration of a simulation of Tic-Tac-Toe. Rewards being added when crosses wins, subtracted when noughts wins and no reward will be given when it is a tie.

In order to determine the best action, a tree is made consisting of all possible actions, from which all branches arise, each with its own branch. When the game

is over, rewards are handed out. When crosses has won, that whole branch gets a reward of 10. When crosses has lost, the reward is -10 and when no one has won, nothing will be added or subtracted. MCTS offers a number of advantages over traditional tree search methods. MCTS does not require any strategic or tactical knowledge about the given domain to make reasonable decisions. The algorithm can function effectively without any knowledge of a game, aside from the legal moves and end conditions. This means that a single MCTS implementation can be reused for a number of games with little customization, making MCTS a potential boon for general play.

MCTS performs a tree growth that adapts to the search space. The algorithm visits nodes of interest more frequently and focuses its search time on more relevant parts of the tree. Large combinatorial spaces typically cause problems for standard depth- or width-based search methods, but the nature of MCTS means it will eventually find those moves that seem optimal and focus its search effort on those.

The way in which the MCTS chooses the best action based on the step-by-step plan shown in figure 5.2.

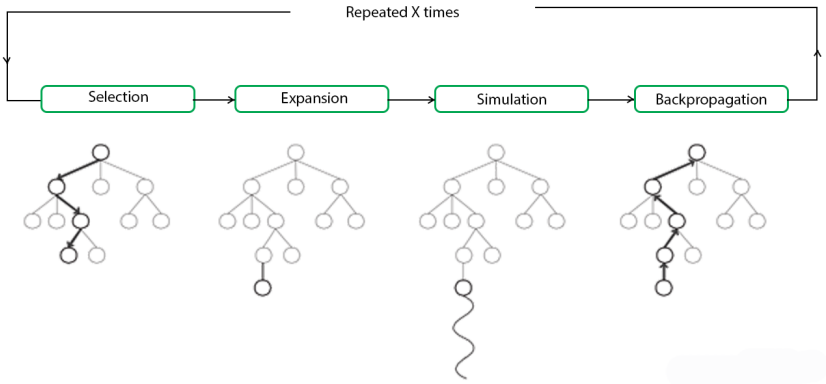

Figure 5.2: Monte Carlo Tree Search roadmap for choosing the best action.

The tree search first starts with a state created by simulating a number of arbitrary nodes. A node represents a possible action. A node can be either a parent or a child. The moment a node is added, the parent and child nodes are moved down. So the lower nodes become the child nodes and the node above it becomes the parent node. Each node consists of two values, the Upper Confidence Bound applied on trees (UCT) and a heuristic value that comes from the neural network. UCT is the factor that decides which node evaluates next in order to maximize the probability of winning the given state. See Figure 5.3.

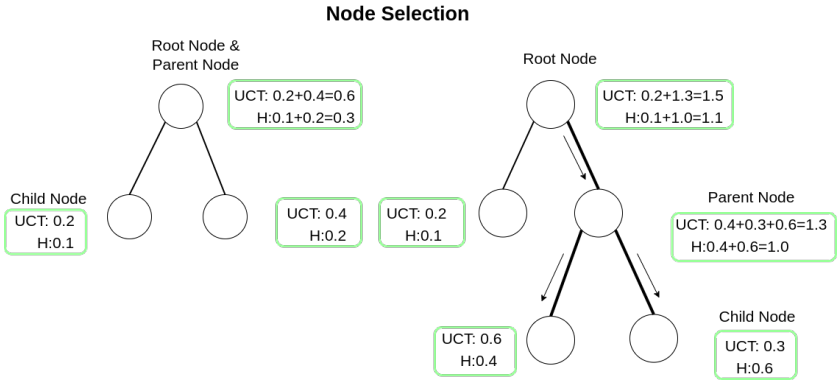

Figure 5.3: Selection of nodes with Upper Bound Confidence applied on trees (UCT) and a heuristic value H given by the neural network.

5.1.1 Selection

The selection is led by the UCT, which treats the problem as the problem of the multi-armed bandits. Imagine being in a casino facing multiple slot machines and each one is configured with an unknown probability of how likely it is that you can get a reward from playing one time. In order to find the best strategy with the highest reward, the dilemma exploration vs exploitation is demonstrated. Exploitation is the task of selecting the move that will lead to the best results so far. Exploration concerns less promising moves that still need to be investigated because of the uncertainty of the evaluation. Suppose one is looking for a new restaurant, it is very probable that one will occasionally eat unpleasant food in these new restaurants. When all the information about the area is known, it is possible to find the best restaurant by even simulating brute force, i.e. by checking every restaurant. The dilemma arises from the incomplete information, it is necessary to gather enough information to make the best decisions while controlling the risk. During exploitation, the most well-known option is taken advantage of. In exploring, some risk is taken to gather information about unknown options. For the calculation of the UCT, the standard formula is stated as:

$$UCT(node_i) = \bar{x} + c\sqrt{\frac{logN}{n_i}} \tag{5.1}$$

Where $\bar{x}$ denotes the mean node value, N number of times the parent has been visited, n_i number of times the child has been visited and c being a constant.

For the optimization of the quantum circuit routing, this formula is structured into the following formula:

$$UCT(node_i) = reward + \sqrt{2}\sqrt{\frac{logN + e + 10^{-}6}{n_i + 10^{-}1}} \tag{5.2}$$

Where $\bar{x}$ represents the reward of that node. The number of times the parent and child have been visited is multiplied by a number of factors to keep the answer within a range of 0 and 1. The term on the left (reward) is the exploitation term. It is simply the average win rate, increasing the better a node has performed in the past. The right term $\sqrt{\frac{logN+e+10^{-}6}{n_i+10^{-}1}}$ is the exploration term. It increases the less

often a node is selected for simulation. The exploration parameter c is a number that can be chosen to determine to what extent the equation prefers exploration over exploitation. The number that is usually chosen is $c = \sqrt{2}$ [15].

The UCT is computed for each node. The tree is simulated up to N number of layers. Suppose N is equal to 3, i.e. as shown in Figure 5.2, the selection part in the tree goes down 3 nodes from the root, with the last nodes representing the leaf nodes. Leaf nodes are the nodes that do not have child nodes themselves. From the leaf node, all UCT are added together to calculate the UCT of the parent nodes. The branch with the highest UCT gets to be picked for the next phase.

To give an idea of how this stage would work, see the pseudo code below:

Algorithm 1 Selection MCTS

```
1: function SELECTION(node)
2:     while node has children do
3:         node = child of node with biggest UCT
       return node
```

5.1.2 Expansion

After the selection phase, the next phase is the expansion phase 5.4.

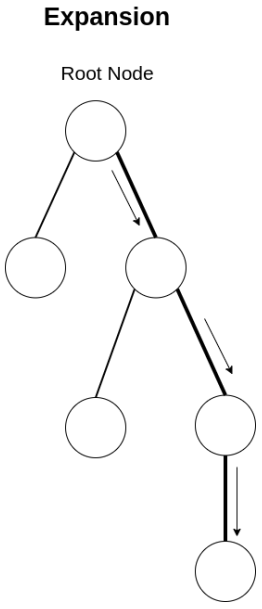

Figure 5.4: The expansion phase of the MCTS

This part is used to further increase the options in the game by expanding the selected node and making many child nodes. Expansion is the strategic task that decides for a given leaf node whether this node will be expanded. The easiest rule is to expand one node per simulated game. If the wing node is already at the end, then the extension part is not needed. See Algorithm 2 for the pseudo code.

Algorithm 2 Expansion MCTS

```
1: function EXPANSION(node)
2:     if node is visited then
3:         generate children of node
```

5.1.3 Simulation

In the simulation phase (also called playout) the best children/leaf node from the group has to be found. The move that performs best and leads to the right answer is determined here. Through the use of Reinforcement Learning (RL), arbitrary decisions can be made in the game further down in any child node.A reward is given to each node of children by calculating how close the output of their random decision was to the final output we need to win the game.

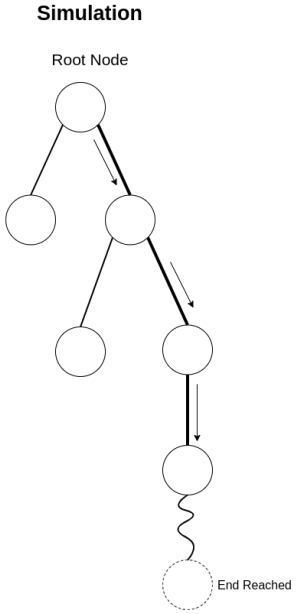

Figure 5.5: The Simulation Phase of the MCTS

For example: In the game of Tic-Tac-Toe. If the random decision to make X next to the previous X in the game results in three consecutive X-X-X, it means that the game is over. The strategic task that selects moves is to play by yourself until the end of the game. This task may consist of playing regular random moves or pseudo-random moves selected according to a simulation strategy. The main idea is to play interesting moves by using patterns, capturing considerations and proximity to the last move. It is the job of the RL model to find out these patterns. When additional lines are added, the condition of a limited number of moves is met. When rules such as: (1) The game is stopped and scored if it exceeds a certain number of moves or (2) the player cannot play until after the opponent has made a move. It is hard to elaborate a efficient simulation strategy.If the strategy is too stochastic, for example if it selects moves almost arbitrarily, then the moves played are often weak. Conversely, if the strategy is too deterministic, e.g. if the move selected for a particular position is almost always the same, because there is too much exploitation taking place, then the exploration of the

search space becomes too selective. The way simulation works is shown in the pseudo code of Algorithm 3.

Algorithm 3 Simulation MCTS

```
1: function SIMULATION(node)
2:     while node state is not final do
3:         get next node random
4:         node = next node
       return node state
```

5.1.4 Backpropagation

Backpropagation is the procedure that propagates the result of a simulated game backwards from the leaf node to the nodes it had to traverse in order to reach this leaf node 5.6. The path chosen by the neural network of the RL model is remembered and once a leaf node has been reached, the same path with the final reward is taken back to pass the actions to the root node so that the game can be played. This reward is counted positively (reward= $+1$) if the game is won, and negatively (reward= -1) if the game is lost. Draws lead to a reward that has not changed (reward= 0). The value of a node is calculated by averaging the results of all simulated games created through this node. However, the best results in the game are obtained by using the common average of the simulations.

After the simulations, the move ultimately played by the program in the actual game is the one that corresponds to the "best child" of the root. There are several ways to determine which child is the best.

- **Max child**: The child that has the highest value.
- **Robust child**: The child with the highest visit count.
- **Robust-max child**: The child with both the highest number of visits and the highest value.
- **Secure child**: The child who maximizes a lower UCT, from all branches with a winning outcome.

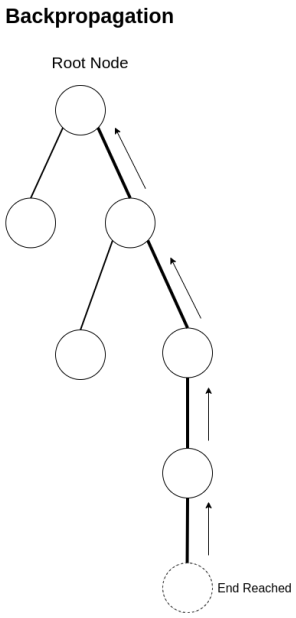

Figure 5.6: The Backpropagating Phase of the MCTS

To give a basic understanding of the backpropagation in MCTS, see Algorithm 4.

Algorithm 4 Backpropagation MCTS

```
1: function BACKPROPAGATION(node)
2:     while node is not null do
3:         increment node score
4:         increment node visits
5:         node = parent node
```

5.2 Implementation of the Reinforcement Learning Model

The MCTS is aided by the use of Reinforcement Learning (RL). RL is a machine learning training method that is based on rewarding desired behavior and/or punishing undesirable behavior. In general, a reinforcement learning model is able to perceive and interpret its environment, take actions and learn through trial and error. The RL will be the potential decision-maker in the MCTS. The goal of a RL algorithm is to find a strategy that will generate the optimal outcome. The difference between RL and supervised machine learning is that the RL model has no labeled data, it has to learn from experience. The way RL accomplishes this goal is by allowing a piece of software called an agent to explore, interact with, and learn from the environment. See Figure 5.7.

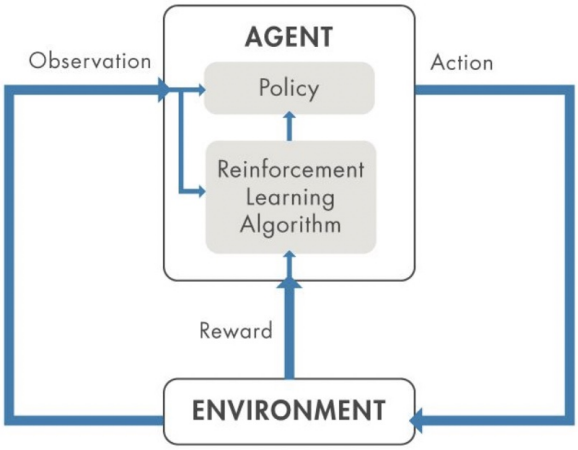

Figure 5.7: Reinforcement learning overview diagram.

Take as an example the maze in Figure 5.8. In order for an RL model to learn to navigate its way out of a maze, it needs a number of defined attributes: an environment, state (observation), reward, and action. In this example, the environment is the maze itself. The state is where the agent is located at the moment. Reward is what the agent receives depending on its action. In the cases of Figure 5.8 a, b and c, the rewards will be negative. The moment the agent has found the way out of the maze, Figure 5.7 d, the agent will receive a high reward. When the agent still has room left for actions, a positive reward will be given, so that the agent knows that it is doing well. Based on the rewards, the agent will adjust its policy to come up with a new tactic. The model learns based on a neural network that falls under the block Reinforcement Learning Algorithm in Figure 5.7.

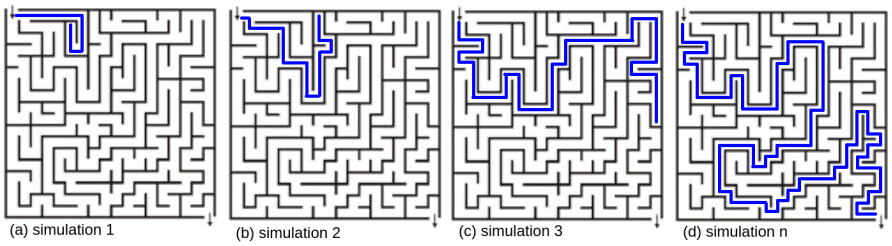

Figure 5.8: Example of reinforcement learning solving a maze.

The neural network generates an action based on the status and reward it receives as input. The neural network trains the agent to deliver better outcomes. In the case of the maze, the actions can be to go up, down, left or right. The neural network outputs a probability for each of these actions based on what it things is the best action, see Figure 5.9. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of an individual node is above the specified threshold, that node is activated and data is sent to the next layer of the network. Otherwise, no data will be passed to the next layer of the network.

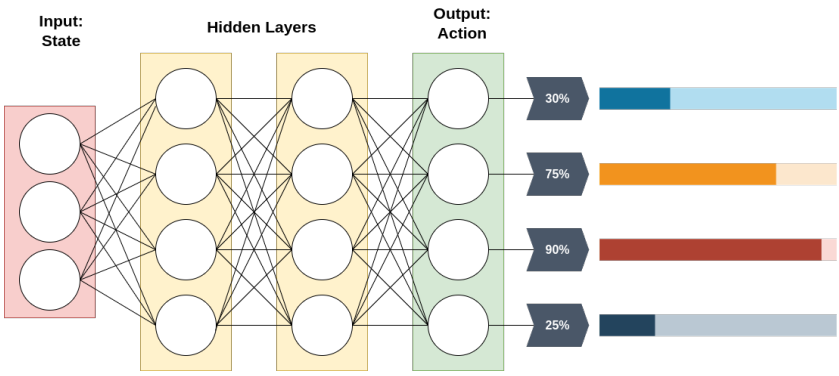

Figure 5.9: Example of a fully connected neural network.

5.2.1 Neural Network Explained

Think of each individual node as its own model, made up of input data, weights, a bias (or threshold), and an output. The formula would look like this:

$$\sum_{i=1}^{m} w_i x_i + bias = w_1 x_1 + w_2 x_2 + w_3 x_3 + bias \tag{5.3}$$

where w stands for the weights and x for an individual node. Weights are assigned once an input layer has been determined. These weights help determine the importance of a particular variable, with larger ones contributing more significantly to the output compared to other inputs. All entries are then multiplied by their respective weights and then added together. After that, the output is passed through an activation function that determines the output. When that output exceeds a certain threshold, it activates the node and passes data on to the next layer in the network. As a result, the output of one node becomes the input of the next node. This process of passing data from one layer to another layer defines this neural network as a feedforward network.

To get a picture of what a single node might look like using binary values, a more tangible example would be to check whether someone should go surfing (Yes: 1, No: 0). The decision to go or not to go is the predicted outcome. Assuming there are three factors that influence the decision making [16]:

- Are the waves good? (Yes: 1, No: 0)
- Is the lineup empty? (Yes: 1, No: 0)
- Has there been a recent shark attack? (Yes: 0, No: 1)

Assuming the following inputs:

- X1 = 1, since the waves are high
- X2 = 0, since it is crowded
- X3 = 1, since there has not been a recent shark attack

Weights are assigned to determine the importance of inputs. Larger weights indicate that certain variables are more important for the decision or outcome.

- W1 = 5, since large waves do not come around often
- W2 = 2, since one is used to the crowds
- W3 = 4, since one may have a fear of sharks

Taking a threshold of 3 as an example, which would translate to a bias value of –3, all the different inputs can start entering values into the formula to get the desired output.

$$prediction = (1 * 5) + (0 * 2) + (1 * 4)3 = 6 \tag{5.4}$$

Using the activation function from the beginning of this section, one can determine that the output of this node would be 1, since 6 is greater than 0. In this instance, going surfing would be predicted, but when adjusting the weights or the threshold, different outcomes are achieved from the model. By observing one decision, as in the example above, it becomes clear how a neural network can make increasingly complex decisions depending on the output of previous decisions or layers.

5.3 Training of the Model

Before the MCTS is used in quantum circuit routing, the circuit is retrieved to see which gates are already operable, with respect to the topology. The moment a gate is not operable, it is handed over to the MCTS together with the gates that are already operable at the moment. In the example of Tic-Tac-Toe 5.1,the whole game board with the actions already filled, represent the root node. In this case part of the circuit is the root node, the gates that are operable up to the first gate encountered that is not operable. Figure 5.10 shows an example of a circuit that wants to operate on linear topology. Assume a sequence qubit allocation (Q0-Q1-Q2-Q3).

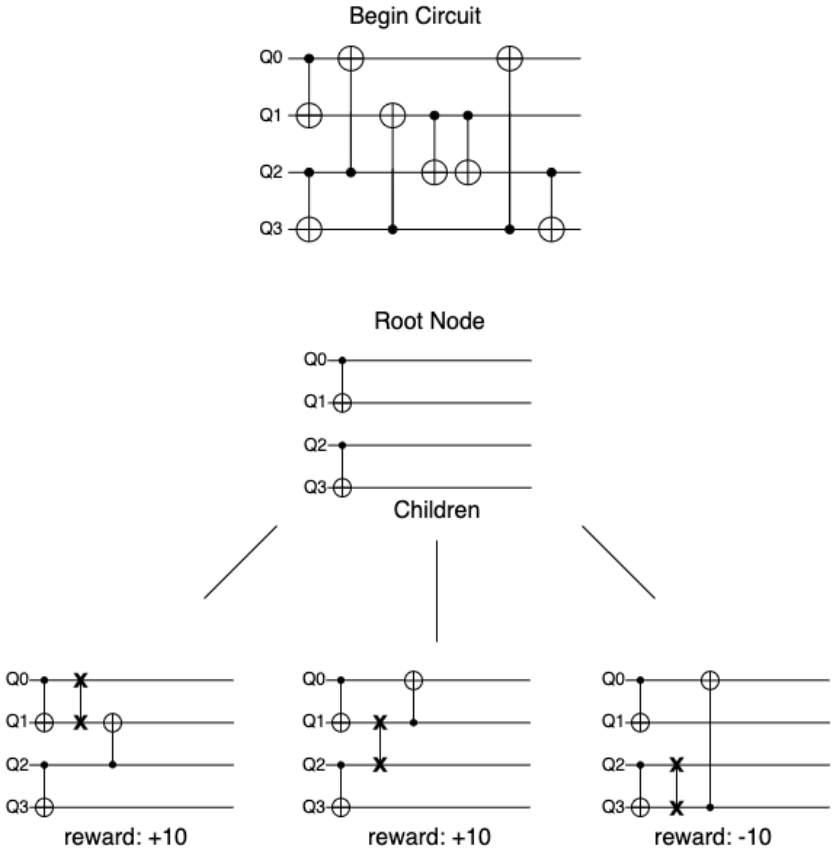

Figure 5.10: MCTS Selection phase on quantum circuit

The child nodes represent the possible locations of the SWAP-gate. The SWAP-gate also transforms the CNOT-gate that was not operable. By looking at the location of the transformed CNOT-gate, it is possible to determine whether it is operable with respect to the topology. The reward can be determined by considering the distance the CNOT gate still needs until it becomes operable. In the case of Figure 5.10, CNOT-gate (2.0) must shift 1 qubit before applying the MCTS to become operable. The left child has moved up the CNOT 1 qubit, making it operable. This has a positive reward for that child. The same goes for the middle child. The right child has caused the CNOT-gate to shift 2 qubits to become operable. Because the situation has deteriorated, this child receives a negative reward. Based on these rewards, the UCT can be calculated and a child can be chosen. In this case, there is a good chance that UCT of the left child and the middle child are the same. A random choice has to be made between the two, because the end of the MCTS has already been reached.

The child that has been chosen is entered into the circuit. The SWAP-gate ensures that gates that interact with the qubits to which the SWAP-gate is applied also change position. It is then further checked whether the upcoming gates are operable. If not, the MCTS is applied and the gates that are already operable so far are fed into the circuit and displayed as the root node. From here the MCTS starts all over again. When a circuit does not consist of 4 qubits, but for example 16, the use of the MCTS with RL is useful. All of the MCTS stages can be run through and the RL model can help with the simulation part. The RL makes choosing a child node faster by assigning a heuristic value to each node. This

heuristic value is a quality value given by the neural network. The node with the highest heuristic value is chosen. The moment the neural network delivers an action, where for example 8 SWAP-gates have been entered in succession, the environment will say that this is too much, for which the agent has to adjust its policy to return better results. By repeating this, the neural network is trained. By executing the MCTS several times and saving the results, they can again be used to train the neural network.

See Figure 5.11 for an understanding of the workflow.

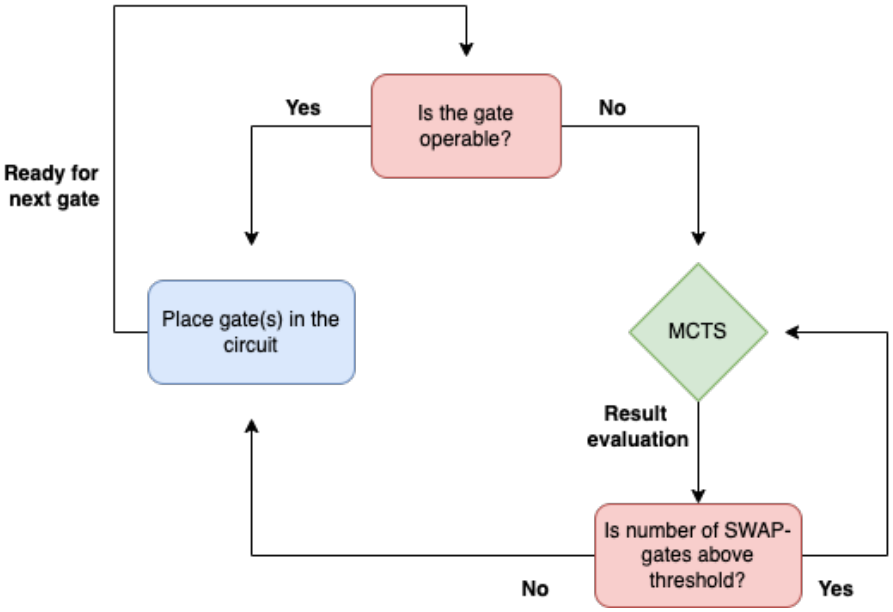

Figure 5.11: MCTS Selection phase on quantum circuit

Because the MCTS contains all possible options, the minimum number of SWAP-gates can be found there. By teaching an RL model to quickly make optimal choices, it can be ensured that the best combination of SWAP-gates will be found. The best combination in this case is a low number of SWAP-gates. The results of this implementation are presented in the next chapter.

Chapter 6

Results

This chapter discusses the results obtained from the MCTS with a benchmark against Qiskit and Cambridge's Tket in combination with Google's Cirq.

6.1 Results of the MCTS

Before training the RL model, 400 simulations have been done with a bare MCTS algorithm. From this, the states (root nodes) and chosen actions that belong to them are stored that the MCTS has faced. Subsequently, the neural network is trained on this. See Figure 6.1 for the fully connected neural network used for this research.

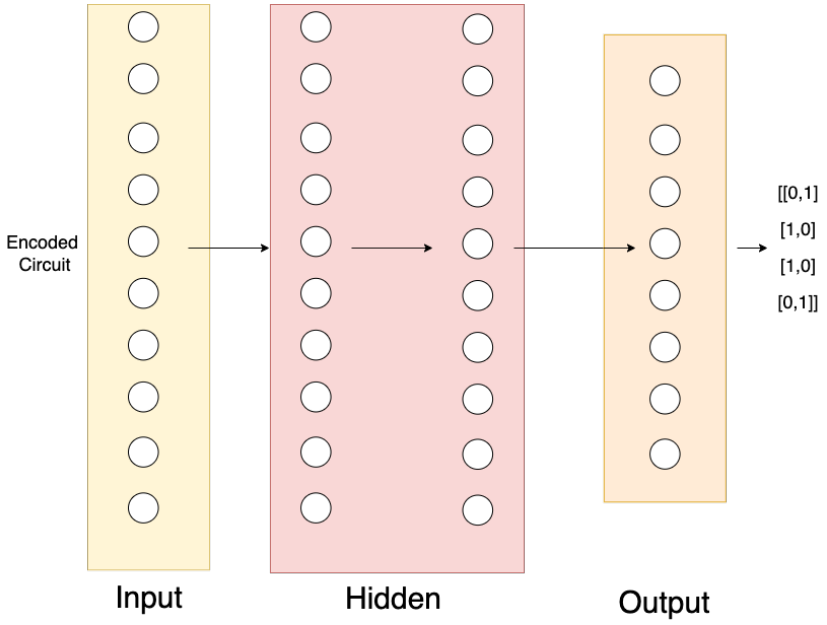

Figure 6.1: Illustration on the Sequential Fully Connected Neural Network used for this research.

The model has been trained based on random qubit allocations and random circuits. In Figure 6.2 the loss of the function can be seen. This figure shows how many times the model has mispredicted. The model has been trained on 110 epochs.

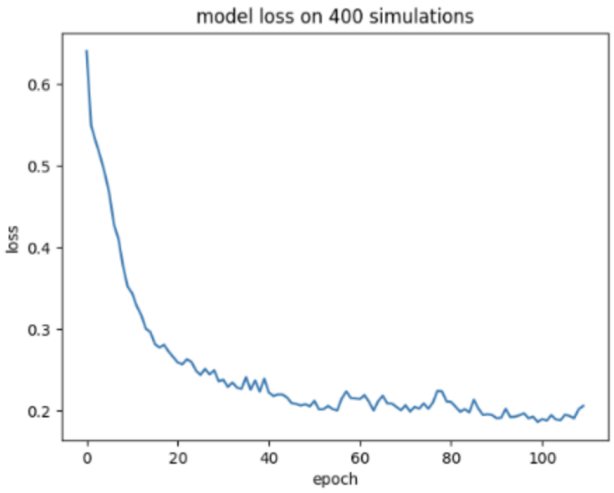

Figure 6.2: Model loss after training on 400 MCTS simulations

This trained model has been applied in the MCTS, where a test circuit has been used to evaluate the results. Take the topology with the random allocation of Figure 6.3 as an example.

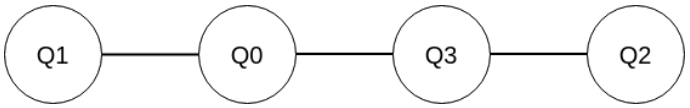

Figure 6.3: Random Qubit Allocation in Linear Topology

The circuit in Figure 6.4 has been used for the evaluation.

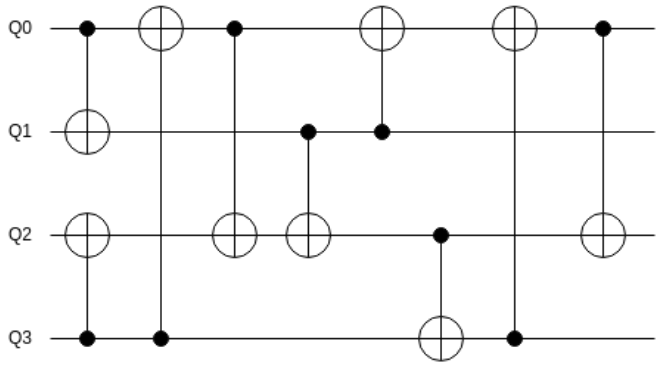

Figure 6.4: Inoperable Test Circuit

Various simulations have been done with this circuit. Out of 20 simulations the circuit with the fewest SWAP-gates was the circuit in Figure 6.5.

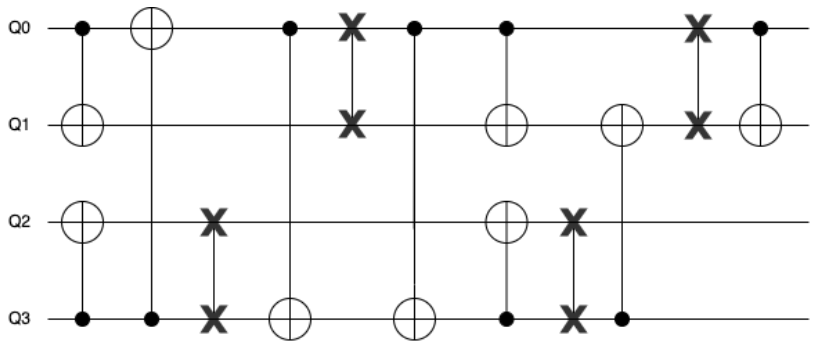

Figure 6.5: Circuit optimized with random qubit allocation

This figure shows that the SWAP-gates ensure that the circuit becomes compatible with respect to the topology. The SWAP-gates have caused the circuit depth to go from 7 to 11.

6.2 Benchmarking of the Circuit Depth

In order to compare the MCTS algorithm with the state-of-the-art, it was decided to compile 5 different circuits 50 times each and compare them with 50 compilations on the Qiskit algorithm and Tket algorithm. An graph on the difference in circuit depths of before and after adding SWAP-gates has been plot for each algorithm. See Figure 6.6 for the MCTS results. This graph shows that there is a clear

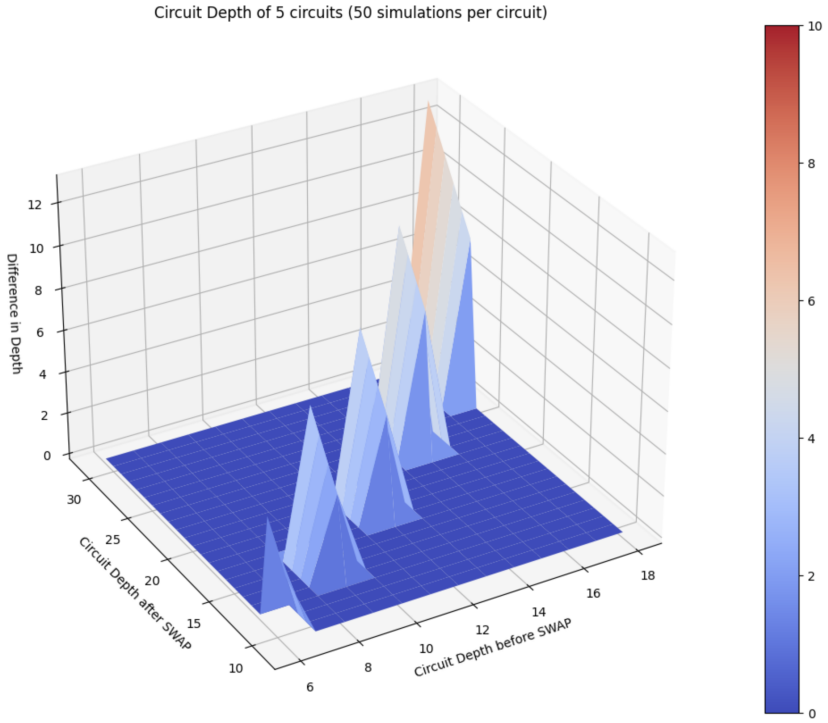

Figure 6.6: MCTS Circuit Depth Results

increase in SWAP gates when the initial circuit depth is also increased. Looking at the values on the z-axis, it can be seen that the values vary enormously, because there is a lot of transition in the colors.

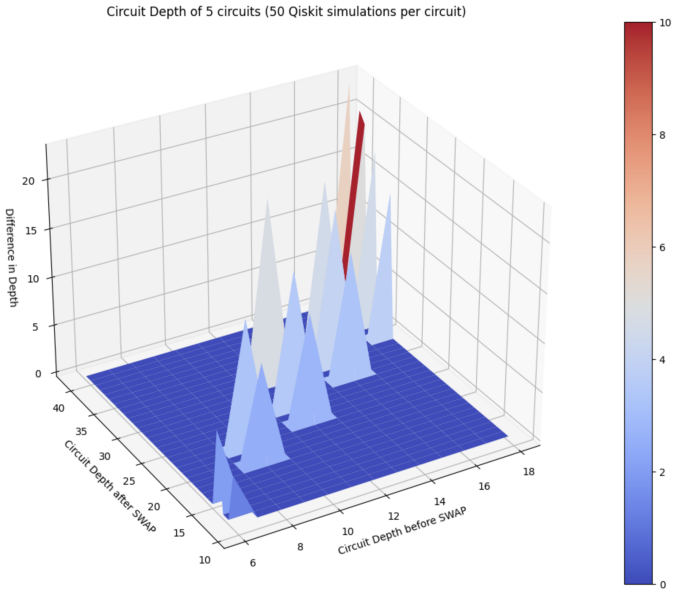

Figure 6.7: Qiskit Circuit Depth Result

The results in Figure 6.7 show the same simulation that has been done, but with the Qiskit algorithm. For this simulation it has been chosen to disable the function 'optimization_level'. This is because this level takes extra steps in addition to adding SWAP-gates to optimize routing. It would make the comparison unfair. The graph shows that the values are more stable, because there are not so many transitions in the difference in circuit depth. Looking at the z-axis, it also shows that the maximum difference in circuit depth is higher than that of the MCTS.

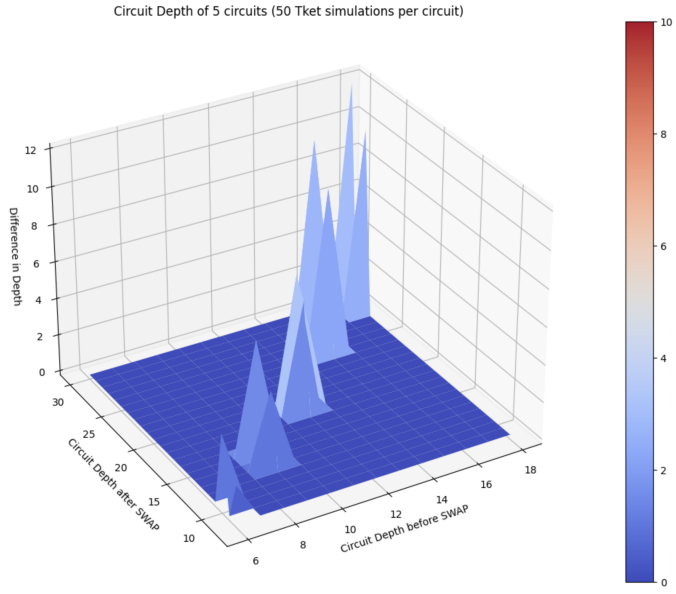

Figure 6.8: Tket Circuit Depth Result

The graph in Figure 6.8 shows the same simulation done with Tket and Cirq. The graph shows that these results are also more stable than those of the MCTS. The difference in depth appears to be lower than that of Qiskit.

Chapter 7

Discussion

In this thesis research has been done on quantum circuit routing optimization by asking the main question 'How can quantum circuit routing be optimized using Reinforcement Learning applied on Monte Carlo Tree Search?'. In order to answer this question, the issues on quantum circuit routing have been highlighted. Another thing that has been looked at is what state-of-the-art approaches already exist to work around these issues. Research has been done in how initial qubit placement can provide a minimized number of SWAP-gates and finally, two state-of-the-art approaches were combined to see how they can help optimize quantum circuit routing.

7.1 Key Findings of the Results

The results showed that MCTS showed varying results. Obviously results from these circuits have been found, where the minimum circuit depth values were lower than those of Qiskit and Tket. The algorithms of Qiskit and Tket show that the difference in circuit depth was more stable than that of the MCTS. It was also seen that Tket's algorithm shows better results than Qiskit's when Qiskit's 'optimization_level' function is turned off.

7.1.1 Interpretation of the Results

The results seem to relate to the previous studies, because not only was the circuit operable ensured, but an optimization in the number of SWAP-gates was also achieved. This has been the goal of most of the researches. The way these results contribute to the field of Quantum Computing is by making it possible to make circuits operable on any topology. A circuit can initially be large in depth and the algorithm may find a minimum number of SWAP gates.

7.2 Limitation of the Study

During the research, some limitations were detected. Unfortunately, it was not possible to implement the initial qubit placement in the algorithm, despite the idea being fully written out. Also, the results could have been more stable if the

RL model had been trained better. Different topologies and differences in the number of qubits in a circuit could have been looked at. If there had been more time for this research, memory efficiency and optimization in runtime could also be looked into.

7.3 Recommendation and Future Work

These results can be a stepping stone for a next researcher who does optimization on quantum circuit routing with the MCTS, RL and an initial qubit placement. This research can be the start of a new state-of-the-art approach, once the ideas described in this thesis are fully implemented. For future work, the initial qubit placement should definitely be implemented and a well-trained neural network should be applied. By training an RL model on different topologies and circuits consisting of several qubits, the neural network can obtain results faster and ensure higher accuracy. Higher accuracy will make the results more stable. Something also described by the researcher Matteo Pozzi, who was interviewed [5**?**] for this research, is runtime optimization. The moment a circuit consists of several qubits, the solution space will be larger, which only increases the runtime. By making the algorithm as efficient as possible, it may be possible to ensure that runtime does not have to be a major issue.

Chapter 8

Conclusion

In this thesis, the question is asked 'How can quantum circuit routing be optimized using Reinforcement Learning applied on Monte Carlo Tree Search?'. From the first chapter it became clear that the problem with quantum circuit routing is that circuits are inoperable with certain topologies. It also became clear that circuits must be small in depth, due to the decoherence of qubits.

Next, looking at what has already been done by previous studies to optimize quantum circuit routing, the state-of-the-art approaches that were most successful were the MCTS approach and the Deep RL approach in combination with the initial qubit placement procedure. Those showed during the benchmark the top results.

Next, research has been done on how to implement this initial qubit placement procedure to minimize the SWAP-gates. Despite being an NP-complete problem, determining a qubit mapping is necessary to ensure that the circuit needs as few SWAP-gates as possible.

Finally, a way in which the MCTS can be used to perform quantum circuit routing, where the number of SWAP gates is as few as possible, was looked at. By having the MCTS guided by a trained RL model, it can be ensured through trial-and-error that the model is trained in such a way that it can find a shortcut in the MCTS using as few SWAP gates as possible as an action .

Bibliography

[1] B. Tan and J. Cong, "Optimality study of existing quantum computing layout synthesis tools," *IEEE Transactions on Computers*, vol. 70, pp. 1363–1373, 2021. (document), 2.4

[2] "Quantum computing transpiler documentation," May 2021. [Online]. Available: https://qiskit.org/documentation/apidoc/transpiler.html (document), 3.2

[3] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for nisq ... - arxiv.org," May 2019. [Online]. Available: https://arxiv.org/pdf/1809.02573.pdf (document), 3.2, 3.3

[4] F. Bova, A. Goldfarb, and R. Melko, "Quantum computing is coming. what can it do?" Jul 2021. [Online]. Available: https://hbr.org/2021/07/quantum-computing-is-coming-what-can-it-do 1.1

[5] M. G. Pozzi, S. J. Herbert, A. Sengupta, and R. D. Mullins, "Using reinforcement learning to perform qubit routing in quantum compilers," Jul 2020. [Online]. Available: https://arxiv.org/abs/2007.15957 1.4, 3.3, 7.3

[6] A. Sinha, U. Azad, and H. Singh, "Qubit routing using graph neural network aided monte carlo tree search," Mar 2022. [Online]. Available: https://arxiv.org/abs/2104.01992 1.4, 3.4

[7] C. Gronlund, M. Li, and S. Lopez, "Dirac-notatie - azure quantum," Apr 2022. [Online]. Available: https://docs.microsoft.com/nl-nl/azure/quantum/concepts-dirac-notation 2.2

[8] D. Shaw, "Quantum hardware - into the quantum jungle," Jul 2020. [Online]. Available: https://www.factbasedinsight.com/quantum-hardware-into-the-quantum-jungle/ 2.4

[9] "Cambridge quantum releases tket version 0.7 with open access to all python users," Nov 2021. [Online]. Available: https://cambridgequantum.com/cambridge-quantum-releases-tket-v0-7/ 3.1

[10] A. Cowtan, S. Dilkes, R. Duncan, A. Krajenbrink, W. Simmons, and S. Sivarajah, "On the qubit routing problem - dagstuhl," Feb 2019. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2019/10397/pdf/LIPIcs-TQC-2019-5.pdf 3.1

[11] S. Herbert and A. Sengupta, "Using reinforcement learning to find efficient qubit routing policies for deployment in near-term quantum computers," 12 2018. 3.3

[12] M. Siraichi, V. F. Dos Santos, C. Collange, and F. M. Quintão Pereira, "Qubit allocation," Feb 2018. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01655951/file/Siraichi_QubitAllocation_CGO18.pdf 4.1

[13] M. Y. Siraichi, V. Fernandes Dos Santos, C. Collange, and F. Magno Quintão Pereira, "Qubit allocation as a combination of subgraph isomorphism and token swapping," Oct 2019. [Online]. Available: https://homepages.dcc.ufmg.br/~fernando/publications/papers/OOPSLA19.pdf 4.4

[14] "Alphago." [Online]. Available: https://www.deepmind.com/research/highlighted-research/alphago 5.1

[15] M. Lu, "General game-playing with monte carlo tree search," Oct 2017. [Online]. Available: https://medium.com/@quasimik/monte-carlo-tree-search-applied-to-letterpress-34f41c86e238 5.1.1

[16] I. C. Education, "What are neural networks?" [Online]. Available: https://www.ibm.com/cloud/learn/neural-networks 5.2.1

[17] J. Preskill, "Quantum Computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, Aug. 2018. [Online]. Available: https://doi.org/10.22331/q-2018-08-06-79

[18] F. Admin, "Qubit dashboard," Jul 2020. [Online]. Available: https://www.factbasedinsight.com/qubit-dashboard/

[19] J. Hui, "Monte carlo tree search (mcts) in alphago zero," May 2018. [Online]. Available: https://jonathan-hui.medium.com/monte-carlo-tree-search-mcts-in-alphago-zero-8a403588276a

[20] D. Silver, "Alphago zero: Starting from scratch," Oct 2017. [Online]. Available: https://www.deepmind.com/blog/alphago-zero-starting-from-scratch

[21] G. Nannicini, L. S. Bishop, O. Gunluk, and P. Jurcevic, "Optimal qubit assignment and routing via integer programming," Jul 2021. [Online]. Available: https://arxiv.org/abs/2106.06446

[22] "Cambridge quantum computing announces update to t: Ket quantum software development kit," Oct 2020. [Online]. Available: https://www.hpcwire.com/off-the-wire/cambridge-quantum-computing-announces-update-to-tket%E2%9F%A9-quantum-software-development-kit/

[23] A. Cotarelo, V. García-Díaz, E. R. Núñez-Valdez, C. González García, A. Gómez, and J. Chun-Wei Lin, "Improving monte carlo tree search with artificial neural networks without heuristics," Feb 2021. [Online]. Available: https://mdpi-res.com/d_attachment/applsci/applsci-11-02056/article_deploy/applsci-11-02056-v2.pdf?version=1614749006