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

May 13, 2022

Abstract

...partial- couple of sentences about motivation / task

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 problems that regular computers just can't solve.

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 [2]. The quantum computers from nowadays may be able to perform tasks which surpass the capabilities of today's regular computers, but the world is not there yet.

1.2 Problem

One of the barriers of programming a quantum computer is the loss of information in a qubit. Think of a spinning coin with information: either heads of tails. Eventually the coin will stop spinning and land on one of the sides. This is the same for a quantum system. The algorithms that are written for a quantum computer contain noise, which can ensure that the information provided is unreliable. 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. So basically, 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 quantum bits (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. The problem here lies in the connectivity matching. 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 flip. This flip 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. The swap gate is performed by placing 3 CNOT gates one behind the other. This ensures that the circuit becomes larger in depth and therefore also more sensitive to decoherence. So 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, universities of applied sciences, MBO institutions, UMCs 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 including 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 the one of Pozzi, et al. [3], where he used Reinforcement Learning and something called Quantum Annealing to optimize the number of SWAP-gates in a circuit. His 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 is done by Sinha, et al [4], where they managed to minimize the number of SWAP gates for a random qubit allocation using Monte Carlo Tree Search in combination with Reinforcement Learning.

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:

- What is quantum circuit routing?

- What has already been done to successfully accomplish quantum circuit routing?
- How can the initial qubit placement procedure be performed?
- How does the Monte Carlo Tree Search work on optimizing the number of SWAP-gates?

This research paper is organised as follows. Section **??** offers a description of basic knowledge about quantum computing phenomenon and techniques needed for understanding the main question. Section **??** demonstrates what research has already been done by others. Section **??** describes the sub-problem initial qubit placement and how to solve it. Section **??** provides an overview of the experimental setup, which is how the Monte Carlo Tree Search is put together with the Reinforcement Learning. Section **??** shows the results of what the reinforcement learning model delivered. Finally, in Section **??** and **??** 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 explains what a quantum computer is and how it can be programmed. It also explains in detail what the problem is that will be optimized.

2.1 The Qubit

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. An illustration of a bit and a qubit can be find in figure 2.1.

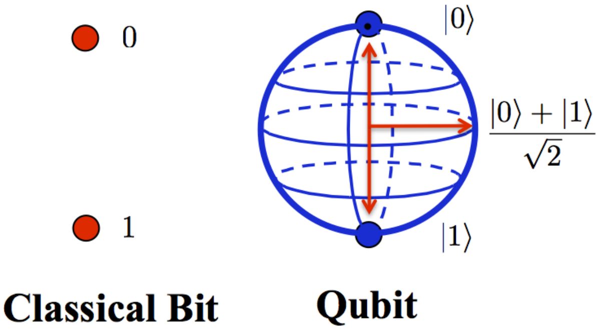


Figure 2.1: Regular bit and quantum bit illustration

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, works 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 a lot in this thesis is the Controlled NOT gate (CNOT-gate), see figure 2.2.

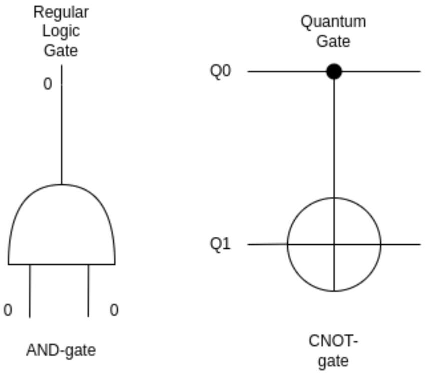


Figure 2.2: Example of a logic gate in a regular computer (left), example of a Quantum gate (right)

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 [5], but in this thesis that is not relevant and the qubits will be represented with a capital Q.

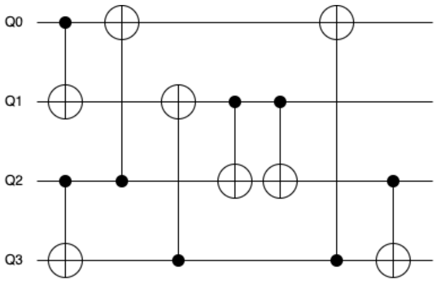


Figure 2.3: A quantum circuit extracted from a programmed script

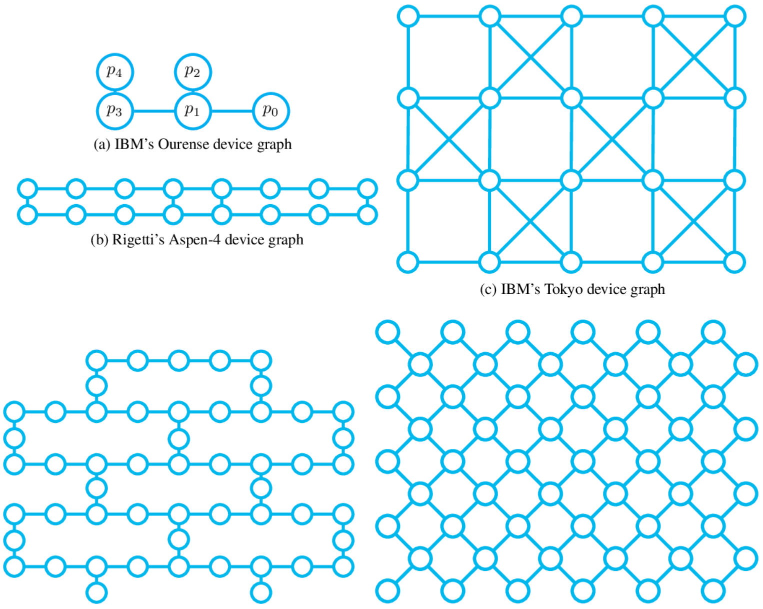
Each horizontal line in the circuit indicates the 'path' of a qubit, each path can have a quantum gate through which the qubit has to pass. Ultimately, the paths can be read to obtain a final result. As mentioned earlier in the example of

Schrödinger's cat, there is a 50% chance that the cat is alive or dead unless the box is put up for checking. 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.



(a) IBM's Ourense device graph

(b) Rigetti's Aspen-4 device graph

(c) IBM's Tokyo device graph

(d) IBM's Rochester device graph

(e) Google's Sycamore device graph

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.

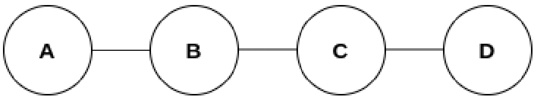


Figure 2.5: Example of a linear topology.

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.

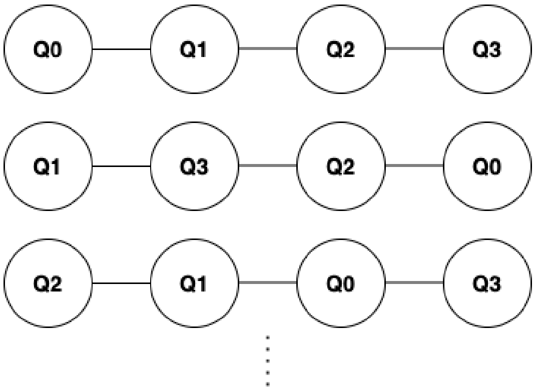


Figure 2.6: Different combinations of qubit mapping.

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.

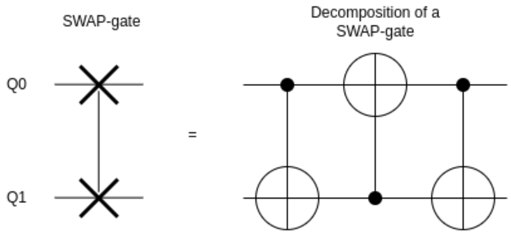


Figure 2.7: A SWAP-gate and its decomposition.

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 [6].

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.

Chapter 3

Previously Researched Work

This chapter will provide a picture of what has been done before to optimize 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. A well-known one is the use of the routing algorithm from Cambridge's Quantum Software Development Kit (Q-SDK) $t\,|ket\rangle$ [7]. The $t\,|ket\rangle$ Q-SDK is designed to maximize the performance of quantum algorithms when running on quantum computing hardware, and to accelerate the development of quantum computing applications across multiple industry sectors. Cambridge students described a solution to the circuit routing problem implemented with $t\,|ket\rangle$ [8]. They believe that this heuristic method in $t\,|ket\rangle$ matches or is better than the results of other circuit mapping systems in terms of depth and total number of gates of the compiled circuit, and has a much shorter run time allowing larger circuits are routed. They represent a quantum computer as a graph in which nodes are physical qubits and edges are the allowed 2-qubit interactions.

3.2 SWAP-based BidiREctional heuristic search algorithm

In the paper written by Gushu Li et al. [9] they describe, a SWAP-based BidiREctional heuristic search algorithm, named SABRE, is proposed to solve this qubit mapping problem and overcome the drawbacks mentioned above. With the observation that many attempts in exhaustive search can be redundant and effective mapping transition needs to start from the qubits in the two-qubit gates that need to be executed, we design an optimized SWAP-based heuristic search scheme in SABRE with significantly reduced search space. They present a novel reserve traversal search technique in SABRE to naturally generate a high-quality 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. Moreover, we introduce a decay effect, which will slightly increase our heuristic cost function values when evaluating overlapped SWAPs, to let SABRE tend to select non-overlapped SWAPs. This optimization enables the control of parallelism in the additional SWAPs and can further generate different hardware-compliant circuits with a trade-off between circuit depth and the number of gates. §

3.3 Integer Linear Program

In the paper written by Giacomo Nannicini, et al. [10] the qubit routing problem is modelled as an integer linear program, using a network flow formulation with binary variables that includes the initial allocation of qubits and their routing. We consider several cost functions: an approximation of the fidelity of the circuit, its total depth, and a measure of cross-talk, all of which can be incorporated in the model. Numerical experiments on synthetic data and different hardware topologies indicate that the error rate and depth can be optimized simultaneously without significant loss. We test our algorithm on a large number of quantum volume circuits, optimizing for error rate and depth; our algorithm significantly reduces the number of CNOTs compared to Qiskit's default transpiler SABRE, and produces circuits that, when executed on hardware, exhibit higher fidelity.

3.4 Reinforcement Learning Approaches

Several approaches have been taken to optimize the circuit routing problem. Most are done through deep reinforcement learning. In Matteo Pozzi's paper [3], it talks about a qubit routing procedure that uses 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 for the qubit routing problem. They mentioned that in order to achieve this, 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.

3.5 Graph Neural Network with Monte Carlo Tree Search

Another approach is to use the Monte Carlo Tree Search (MCTS) guided by a Graph neural network. 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 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.

Chapter 4

Initial Placement of the Qubits

This section introduces the qubit allocation problem and gives a understanding of one of the possible approaches to solve this problem.

4.1 Qubit Allocation

Given a quantum circuit and an architecture, we want to know 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 an allocation 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 Graph Topology. Which means the direction of the edges matters. To illustrate this, assume the topology of Figure 4.1.

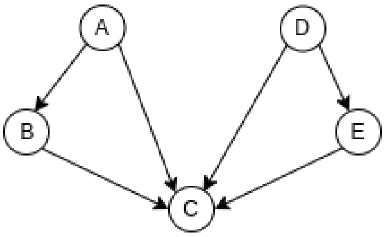


Figure 4.1: Toy topology for the initial qubit placement

And circuit interacting on this topology illustrated in Figure 4.2.

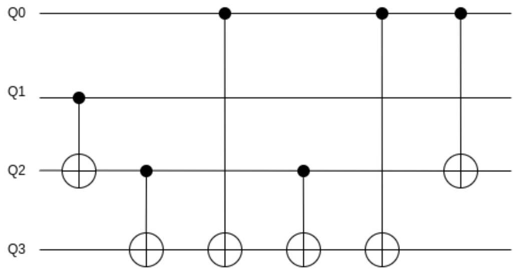


Figure 4.2: Toy circuit model of CNOT gates for the initial qubit placement

4.2 Weighted Directed 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 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 Graph 4.3 can be made, illustrating the operations that need to be performed.

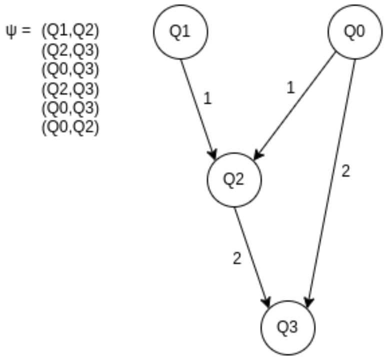


Figure 4.3: Coupling Directed Graph of the toy circuit model

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 allocate 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 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.

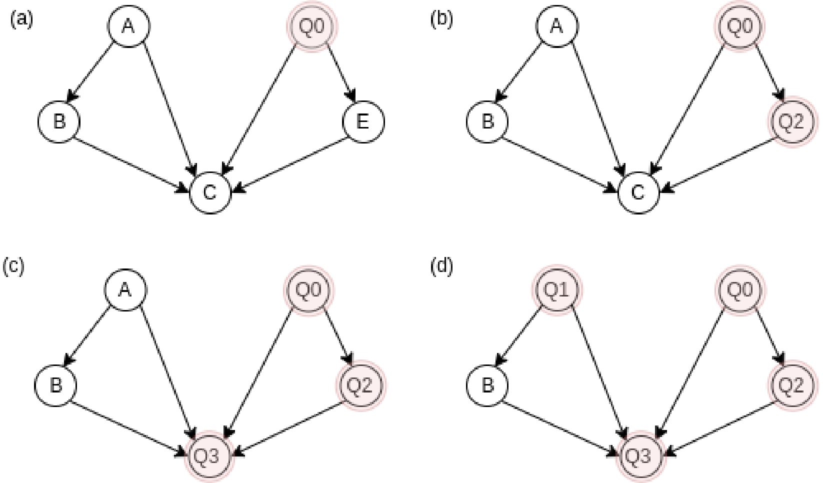


Figure 4.4: talk about how we start with the highest out-degree and match them on the highest out-degree of the actual topology, we do this recursively. starting from (a) and going from there.

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 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.

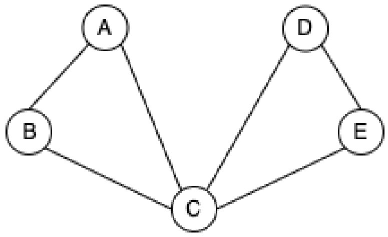


Figure 4.5: Undirected Graph Topology

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 allocated in 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 allocating 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 allocation of the logical qubits.

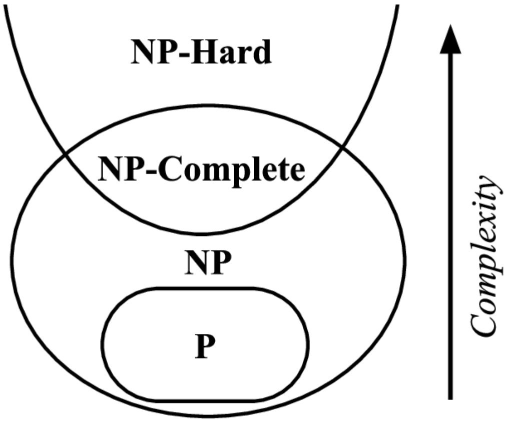


Figure 4.6: Nondeterministic Polynomial time complexity classes

In Figure 4.7 the same sketch is made of a way in which the qubits can be allocated 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.

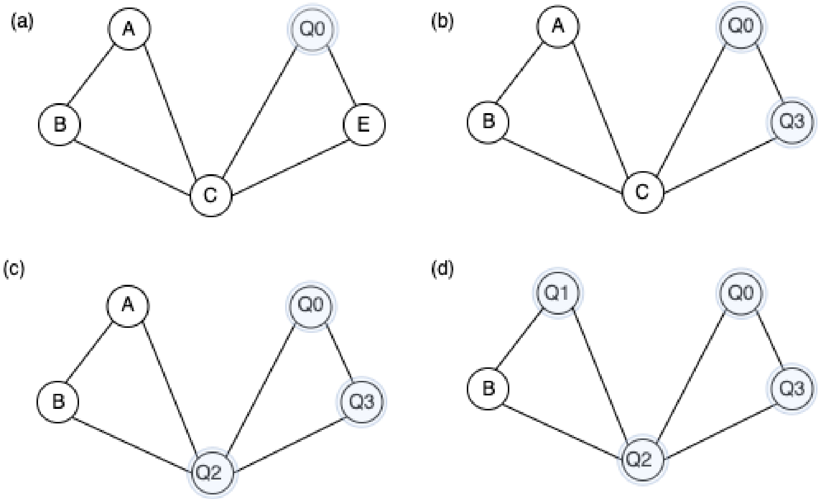


Figure 4.7

Chapter 5

Monte Carlo Tree Search with Reinforcement Learning

This chapter will describe how the Monte Carlo Tree Search (MCTS) in combination with Reinforcement Learning (RL) has been applied to optimize quantum circuit routing.

5.1 Implementation of the MCTS

In 2016, the computer program AlphaGo was nominated as the breakthrough of the year. The program consists of a combination of advanced MCTS with deep neural networks and is made to play the Asian game Go [13]. The idea for using the MCTS for the quantum circuit routing is inspired by AlphaGo, where all possible actions are simulated, until a best answer comes out. MCTS is used to determine subsequent 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.

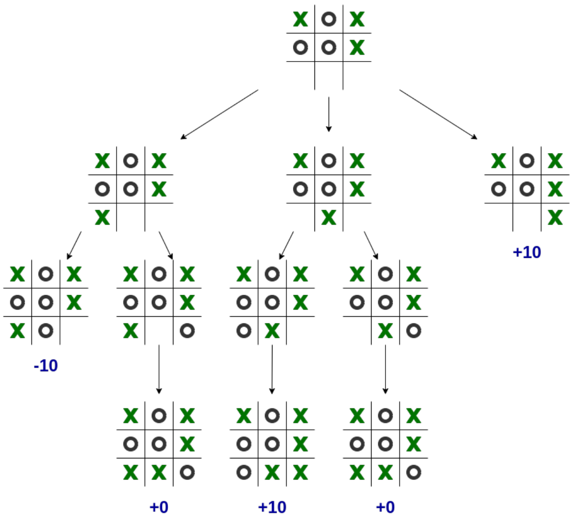


Figure 5.1: Simple illustration of simulation of Tic-Tac-Toe. Rewards being add when crosses wins, subtracted when noughts wins and no reward will be given when it is a tie.

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 given 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 with no knowledge of a game apart from its legal moves and end conditions; this means that a single MCTS implementation can be reused for a number of games with little modification, and makes MCTS a potential boon for general game playing. MCTS performs asymmetric tree growth that adapts to the topology of the search space. The algorithm visits more interesting nodes more often, and focusses its search time in more relevant parts of the tree. This makes MCTS suitable for games with large branching factors such as 19x19 Go. Such large combinatorial spaces typically cause problems for standard depth- or breadth-based search methods, but the adaptive nature of MCTS means that it will (eventually) find those moves that appear optimal and focus its search effort there.

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

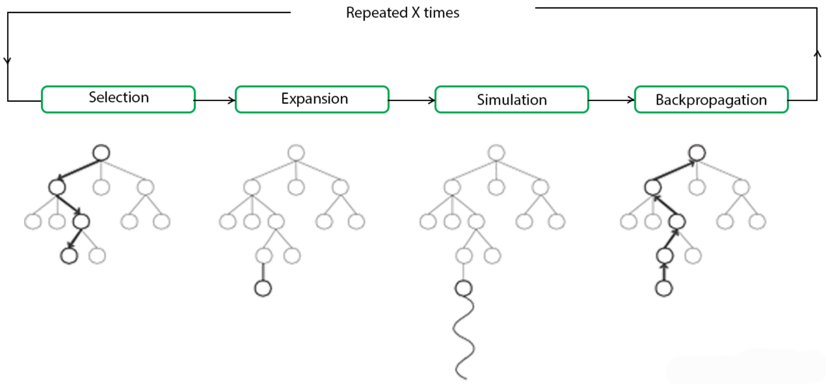


Figure 5.2: Monte Carlo Tree Search roadmap for choosing the best action.

The tree search first starts with a state that by simulating a number of random nodes. A node represents a possible action. A node can be a parent or a child. The moment a node is added, the parent and child nodes move down. So the bottom 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 evaluate next in order to maximize probability of victory from the given state. 5.3.

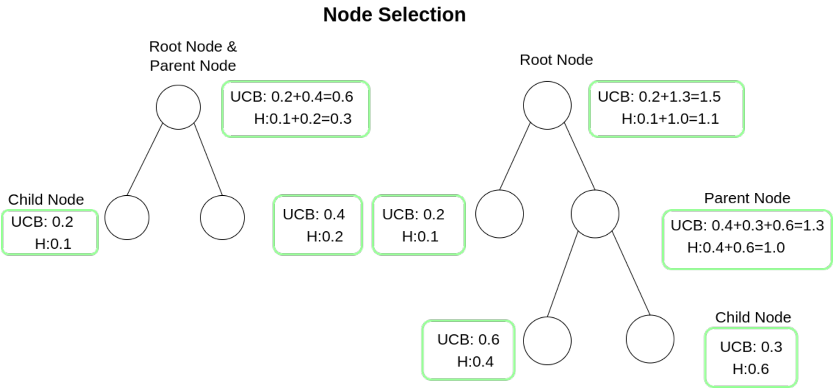


Figure 5.3: Selection of nodes with Upper Bound Confidence (UCB) applied on trees and a heuristic value H given by the neural network.

5.1.1 Selection

The selection is guided by the UCT, which treats the problem as the multi-armed bandit problem. Imagine you are in a casino facing multiple slot machines and each is configured with an unknown probability of how likely you can get a reward at one play. To figure out the best strategy with highest reward, the exploration vs exploitation dilemma is demonstrated. Exploitation is the task to select the move that leads to the best results so far. Exploration deals with less promising moves that still have to be examined, due to the uncertainty of the evaluation. Say, one is looking for a new restaurant, it is very likely to eat unpleasant food from time to time at these new restaurants. If all the information about the environment is known, it is possible to find the best restaurant by even just simulating brute-force, so by checking every restaurant. The dilemma comes from the incomplete information, it is necessary to gather enough information to make the best decisions while keeping the risk under control. With exploitation, advantage is taken from the best known option. When exploring, we take some risk to gather information about unknown options. For calculating the UCT, the standard formula is stated as:

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

Where $\bar{x}$ denoted 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 quantum circuit routing optimization, this formula has been built up to the following formula:

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

Where $\bar{x}$ is depicted as the reward of that node. The number of times the parent and child have been visited is multiplied by some factors, to keep the answer within a range of 0 and 1. The left term (reward) is the exploitation term. It is simply the average win rate, going larger the better a node has historically performed. The right term $\sqrt{\frac{logN+e+10^-6}{n_i+10^-1}}$ is the exploration term. It goes larger the less frequently a node is selected for simulation. The exploration parameter c is just a number we can choose that allows us to control how much the equation favors exploration over exploitation; the usual number chosen is $c = \sqrt{2}$ [14].

The UCT is calculated 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 UCTs are added together to calculate the UCT of the parent nodes. The branch with the highest UCB gets to be picked for the next phase.

5.1.2 Expansion

After the selecting phase, the next phase is the expanding phase. This part is used to increase the options further in the game by expanding the selected node and creating many children nodes. Expansion is the strategic task that, for a given leaf node, decides whether this node will be expanded by storing one or more of its children in memory. The simplest rule is to expand one node per simulated game. The expanded node corresponds to the first position encountered that was not stored yet. Note that the expansion could also be performed after the simulation.

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 which will perform best and lead to the correct answer down the tree is decided here. Through the use of Reinforcement Learning (RL) random decisions can be made in the game further down from every children node. Then, a reward is given to every children node by calculating how close the output of their random decision was from the final output that we need to win the game. For example: In the game of Tic-Tac-Toe. If the random decision to make X next to previous X in the game result in three consecutive X-X-X, that means the game has ended. The strategic task that selects moves is through self-play until the end of the game. This task might consist of playing plain random moves or pseudo-random moves chosen according to a simulation strategy. The main idea is to play interesting moves by using patterns, capture considerations, and proximity to the last move. It is the RL model's job to figure out these patterns. When extra rules are added, the condition of limited number of moves is satisfied. When rules as: (1) The game is stopped and scored if it exceeds a given number of moves or (2) the player can only play after the opponent has set a move. Elaborating an efficient simulation strategy is a difficult issue. If the strategy is too stochastic, for instance if it selects moves nearly randomly, then the moves played are often weak. In contrast, if the strategy is too deterministic, for instance if the selected move for a given position is almost always the same, i.e. too much exploitation takes place, then the exploration of the search space becomes too selective.

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 to reach this leaf node. This result is counted positively (reward$= +1$) if the game is won, and negatively (reward$= -1$) if the game is lost. Draws lead to a result reward$=$ 0. The value of a node is computed by taking the average of the results of all simulated games made through this node. However, the best results in game play have been obtained by using the plain average of the simulations.

After the simulations, the move finally played by the program in the actual game is the one corresponding to the "best child" of the root. There are different ways to define which child is the best.

- **Max child**: The max child is the child that has the highest value.
- **Robust child**: The robust child is the child with the highest visit count.
- **Robust-max child**: The robust-max child is the child with both the highest visit count and the highest value. If there is no robust-max child at the moment,more simulations are played until a robust-max child occurs.
- **Secure child**: The secure child is the child that maximizes a lower UCT, which maximizes the quantity.

5.2 Implementation of the Reinforcement Learning Model

5.3 Training of the Model

Chapter 6

Results and Discussion

6.1 Results and Analyses

What did you find?

6.2 Discussion

Did your findings support your hypothesis? Why? Why not?

Chapter 7

Conclusions and Future Work

This Chapter concludes the thesis by summarizing the findings from the study, the contributions and possible limitations of the approach. It can also identify issues that were not solved, or new problems that came up during the work, and suggests possible directions going forward. [**?**]

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] 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

[3] 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, 3.4

[4] 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

[5] 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

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

[7] "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

[8] 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

[9] 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 3.2

[10] 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 3.3

[11] S. Herbert and A. Sengupta, “Using reinforcement learning to find efficient qubit routing policies for deployment in near-term quantum computers,” 12 2018. 3.4

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

[14] 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