6-2022

High Error Rate Qubit Routing

Sean Bonaventure
spb4464@rit.edu

High Error Rate Qubit Routing

Sean Bonaventure

High Error Rate Qubit Routing

Sean Bonaventure

June 2022

A Thesis Submitted
in Partial Fulfillment
of the Requirements for the Degree of
Master of Science
in
Computer Engineering

RIT | Kate Gleason College of Engineering

Department of Computer Engineering

High Error Rate Qubit Routing

Sean Bonaventure

Committee Approval:

Sonia Lopez Alarcon *Advisor* Date
Department of Computer Engineering

Cory Merkek Date
Department of Computer Engineering

Ben Zwickl Date
School of Physics and Astronomy

Acknowledgments

I'd like to thank my Advisor, Dr. Sonia Lopez Alarcon for all her help and working with me through my thesis. I would also like to thank my family for supporting me through my entire educational career.

To my dog, Maggie

Abstract

The current state of quantum computers is characterized by its limited resources and high noise levels. These are known as Noisy Intermediate Scale Quantum Computing (NISQ). Reduction of noise is addressed at the technology level, while noise mitigation strategies are proposed through the compilation process of quantum circuits. The compilation process entails a number of steps that are computationally intensive and scale poorly as the problems grow in size and resource usage. This paper addresses the problems associated with noise by proposing a noise aware qubit routing algorithm. This algorithm attempts to improve accuracy of circuits by using SWAP gates to avoid links with high error rates. This differs from existing algorithms which try to minimize the amount of SWAP gates used. In addition, multiple metrics are evaluated for Qiskit's routing algorithms. The proposed algorithm improves the accuracy against circuits compiled using Qiskit's basic routing algorithm, and against other more sophisticated routing algorithms, depending on the application circuit. Other metrics being considered, the routing algorithm also demonstrates to have minimal computational cost when compared to other approaches. Lastly, it is shown that different circuit benefit from different routing algorithms.

Contents

List of Figures

List of Tables

Acronyms

ALAP

As Late as Possible

ALU

Arithmetic Logic Unit

ASAP

As Soon as Possible

BV

Bernstein-Vaziran

CNOT

Controlled-Not

HERR

High Error Rate Routing

NISQ

Noisy Intermediate Scale Quantum

QFT

Quantum Fourier Transform

SABRE

SWAP-based BidiRectional

Chapter 1

Introduction

1.1 Motivation

Quantum computing has become one of the most promising emerging technologies in recent years. It holds the potential of providing extremely large speedups over classical computers for certain problems. Despite this, there are still many hurdles in the way of efficient quantum computing, most notably high noise rates and low qubit numbers in current computers. Because of these problems the current era of quantum computing has been referred to as Noisy Intermediate Scale Quantum Computing (NISQ).

As the name implies, current quantum computers are so noisy that large scale useful circuits are not yet feasible. IBM's superconducting quantum computer for instance can have error rates 10% [1] or higher for two qubit gates. As such much effort has been put forward to combat noise on quantum systems. This paper focuses on how improvements in the compilation process can be used to reduce the effect of noise and improve circuit accuracy.

The job of the quantum compiler is to translate a quantum circuit into a form that can be executed on a quantum computer. This includes decomposing gates into the quantum computers basis gates, performing optimizations, mapping, routing, and more. This paper focuses specifically on the qubit routing problem. Qubit

routing is a problem resulting from the design of modern quantum computers. In a superconducting qubit system like IBM's computers, not every qubit is able to interact with each other, which creates a problem when a circuit requires a CNOT gate —a two qubit gate— between two qubits that cannot interact. To solve this, SWAP gates are used to swap the states of the argument qubits to a new set that can interact. While this solution guarantees that with enough SWAP gates all qubits can interact, it has its own problems. SWAP gates are expensive and add depth to a quantum circuit. In a noisy system that can cause large errors to accumulate as a result of the added SWAP gates. Therefore routing algorithms can have a great impact on circuit accuracy.

1.2 Contribution

The goal of this work is to develop a quantum routing algorithm that improves the accuracy of a quantum circuit compared to a well established alternative algorithms. Previously, this has been done by creating algorithms that attempt to reduce the amount of gates added during the routing process, but in this work we address the routing problem in a different way - a noise based approach. The contribution of this thesis is therefore:

1. To develop a novel routing algorithm that utilizes noise based routing
2. To benchmark this algorithm to determine its efficacy compared to existing algorithms

1.3 Previous Work

There has been a large amount of work with the focus on addressing the routing problem. Qiskit, IBM's quantum computing frameworks, even offers four different

routing options for their compiler. The main focus of almost all these routing algorithms has been the reductions of added SWAP gates. This is done on the premises that SWAP gates increase depth which can increase the total error of a circuit. There has, however, not been much focus on accounting for the noise itself in the routing process. Not every qubit link in a system has the same accuracy. Some link can have a CNOT gate error rate 10x higher than another. There has been a small amount of work regarding noise aware routing [1] [2]

While the majoirty of the focus in this field has been on reducing the number of CNOT gates added, there have been some works that do focus on noise aware mapping and routing. [1] [2] both address the problem of noise adaptive mapping and routing in similar ways. They address the problem by: creating an ideal initial mapping based off noise, and performing routing using paths with low noise. Both these algorithms are expanded upon in the background section. While these papers both address the mapping and routing problem using noise, they both approach it in different ways than us. Therefore we believe this work is unique.

We address this problem by proposing an algorithm that routes attempts to avoid high error rate qubit links by inserting extraneous SWAP gates to avoid links. We utilize a cost function to determine if the cost of avoiding a link is better than not. This approach solves the routing problem while also improving the overall accuracy of the circuit by avoiding high error rate qubits.

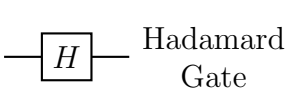
Chapter 2

Background

A gate-based quantum computer performs operations on either one or two qubits at a time. Any operations that require more than two qubits are broken up into multiple two qubit gates. A quantum circuit is a collection of qubits and a sequence of quantum gates that operate on them.

2.1 Quantum Computing

Quantum computing is a technology that attempts to exploit the properties of quantum mechanics to perform computations. The main way this is done is by using superposition and entanglement to encode information and perform operations. That said, there are many different ways to do so. One well established way is a gate based approach. Gate based quantum computing involves using a quantum bit, or qubit, to represent data. These qubits operate in a superposition of 0 or 1. They can be translated to a classical bit (a one or zero), by measuring their value and collapsing their superposition. A quantum computer performs operations on qubits using quantum gates. A quantum gate is an operation that when performed on a qubit modifies its state [3]. A gate-based quantum computer performs operations on either one or two qubits at a time. Any operations that require more than two qubits are broken up into multiple two qubit gates. Figure 2.1 shows an example of a one and two qubit gate. Figure 2.1a shows a one qubit gate, a hadamard gate. The hadamard gate



(a) A hadamard gate, an example of a single qubit gate.

(b) A CNOT gate, an example of a two qubit gate

Figure 2.1: An example of a single and double qubit gate

operates on a single qubit and puts the qubit into a state of super position. Figure 2.1b shows an example of a two qubit gate, a controlled not, or CNOT gate. A CNOT gate operates on two qubits and is the quantum equivelent of a NOT gate. It consists of a control qubit and a target qubit. If the control qubit is a '1', it inverts the state of the target qubit.

A quantum circuit is a collection of qubits and a sequence of quantum gates that operate on them. Figure 2.2 is an example of a simple circuit composed of two hadamard gates and a CNOT gate. This circuit puts both qubits into a state of super position and performs a CNOT operation on them, then measures the result.

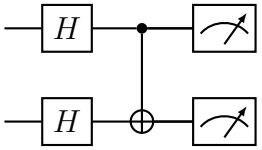


Figure 2.2: An example of a quantum circuit that puts both qubit into a superposition then performs a CNOT

2.1.1 Output of a quantum circuit

In a classical computation the output of an operation is usually represented as a binary string. The calculation 2+2 when performed on an ALU yields the output 0b100 in binary, which represents 4. In a quantum computer the output is obtained by measuring the value of a qubit onto a classical bit. This however represents a problem. If a quantum system is probabilistic, that means the output is probabisitic and not guaranteed to be the same value. For instance, if the state of a qubit is

$\frac{1}{\sqrt{2}}$, the probability it is measured as a 1 is 50%. This is a problem, as in classical computing we are used to running a program once and getting the consitant results. In quantum computing it is necessary to run a circuit many times to collect data and determine a correct result. This is especially true in the noisy era of quantum computing, where noise causes the state of qubits to drift. When analyzing the results of a circuit a histrogram is usually generated to analyse the circuit. An example of an output histogram is shown in Figure 2.3. [3]

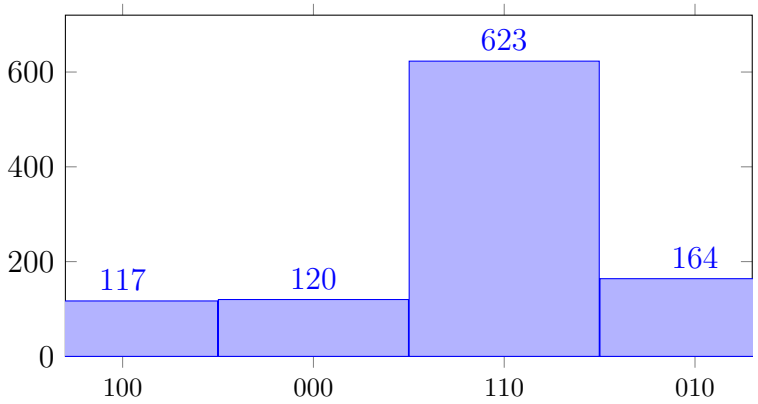


Figure 2.3: Average Accuracy for each routing algorithm with two noise ranges

Figure 2.3 is an example of what an output histogram of a quantum circuit looks like. In this case, the desired output was 110, which is by far the most frequently occurring result. This specific circuit was run 1024 times. In this thesis we refer to accuracy as the $\frac{\#correct}{totaltrials}$. In this instance the correct result was obtained 623 times, making the total accuracy for this circuit 63%.

2.2 Quantum Compilers

The quantum circuit representation shown in Figure 2.2 is a high level representation of how a quantum circuit acts, but in that state is not able to run on a quantum computer. Similar to how a C program cannot be natively run on a computer, neither can a quantum circuit. Like a C program a quantum circuit must be compiled into

a format the quantum computer can recognize and run. This process is called either compilation or transpilation and done by a transpiler or compiler.

A quantum compiler has many responsiblities, but some of its main responsibilities are: decomposing circuits into their basis gates, providing an initial circuit mapping, performing qubit routing, and scheduling.

2.2.1 Decomposing into basis gates

In classical computing a structure such as in if statement cannot be directly run on a processor. It must instead be decomposed into is basis assembly instructions. One of the jobs of a C compiler is to perform this decomposition. A quantum circuit is no different. The hadamard gate shown in Figure 2.1a cannot be directly executed on quantum hardware, but needs to be decomposed into a set of basis gates. Quantum computing theory shows that any quantum operation can be performed with a combination of 3 basis gates. Different quantum computers can each have a different set of basis gates, and its the compilers job to determine what set is needed and how to decompose a circuit. [3]

2.2.2 Mapping and Routing

Another issue presented by the representation of a quantum circuit shown in Figure 2.2 is that it implies a CNOT gate can be performed between any two qubits on a circuit. This is incorrect, as in actual quantum hardware not every qubit can communicate with another. This is referred to as the connectivity of a quantum system.

The connectivity of the qubits in a quantum computer (which qubits can interact with which) is described by the coupling map: a graph in which nodes represents qubits, and edges represent which qubits can interact.

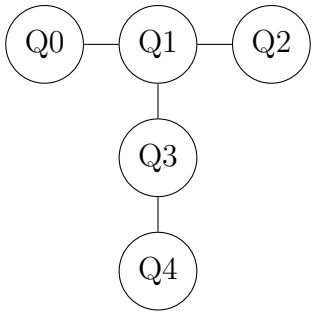


Figure 2.4: Example Coupling graph. Each node represent a qubit and each edge represents the interaction between qubits

Fig. 2.4 shows an example coupling map for the IBM Quito architecture. In this architecture, there are only 5 qubits. Fig. 2.4 shows that Q0 can interact with Q1, but not Q2 as there is not a direct edge connecting them. Therefore if a circuit required a CNOT between Q0 and Q2, a SWAP gate would need to be inserted between Q0 and Q1. By inserting a SWAP gate between the two qubits, the qubits in the circuit no longer match up with the original mapping to qubits on hardware. That is, the state of qubit 0 in the circuit is now on qubit 1 of the hardware. We refer to the state of qubits in the circuit as logical qubit, and will be denoted as "qn". The qubits on hardware are the physical qubits, and will be denoted as "Qn". The relationship between logical and physical qubits is called the **mapping** of qubits.

2.2.3 Scheduling

A quantum circuit can also be thought of as a set of operation that are performed on qubits. Each of these operations has dependencies, and other operations depend on its result. This means there needs to be some sort of order to the operations to ensure the data dependencies are handled correctly. This process is called scheduling usually takes place during the compilation process.The schedule of a quantum circuit can drastically affect the performance of the circuit. Certain scheduling configurations can cause longer dependency chains that force large amounts of dead time, where

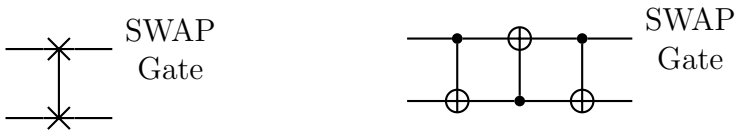
some qubits may not performing calculations. For instance, Itoko et al were able to achieve up to a 7.3% reduction in the execution length of certain quantum circuits by improving the scheduling, relative to Qiskit [4].

Qiskit implements two basic, but very effective and popular forms of scheduling: As Soon As Possible (ASAP) and As Late as Possible (ALAP). In short, ASAP attempts to execute an operation as soon as the schedule allows, while ALAP attempts to execute an instruction as late as it can (usually right before the result of the operation is needed) [5].

2.2.4 Routing and Error

In the example shown in Figure 2.4, a SWAP gate could be inserted between Q0 and Q1. A SWAP gate could also be inserted between Q1 and Q2 for the same effect. The choice of where to insert a SWAP gate is called **routing**. Routing plays a major effect on the accuracy of the circuit. For instance, suppose the next operation in this circuit were a CNOT between q0 and q3. If the aforementioned SWAP was inserted between Q0 and Q1, no additional SWAP would be needed, as Q1 now holds the state of q0 and is connected to Q3. On the other hand, if the SWAP was inserted between Q1 and Q2, another SWAP gate would be needed. This is detrimental to the circuit as a whole since each SWAP gate causes errors to accumulate. Therefore, a routing algorithm that reduces the amount of SWAPs inserted is ideal. This can also be thought of a reducing the **depth** of the circuit. The dept of the circuit is the number of gates in the critical path. A deeper circuit is a circuit that has more gates in a row executing. This increases the total error of the circuit. [6] [1]

Reducing the number of SWAP gates is important due to the accumulation of gate error. Gate errors are accumulated through the execution of gates. Each SWAP gate decomposes into three CNOT gates, and their use can drastically contribute to the computation's overall noise level. Figure 2.5 shows how a SWAP gate is decomposed



(a) A SWAP gate

(b) A SWAP gate decomposed into 3 CNOT gates

Figure 2.5: Decomposition of a SWAP gate

into 3 separate CNOT gates.

The latest advances in superconducting quantum computers have yielded single qubit gate errors averaging around 0.2% , while CNOT gates have an average error rate of 4% [1]. Gate error rates can also vary drastically across the different qubits of a system. A link between one set of qubit could have a CNOT error rate of 2% while another could have an error rate of 10%[1]. IBM publishes calibration data once a day that contains gate error rates and coherence times.

Much work has been done to solve the routing problem effectively, some proposed algorithms were noise-aware [2] [1]. These works focused on routing in such a way that *if* SWAP gates need to be inserted because of non-connected qubits, it is done through a path that minimizes error, even if it requires more SWAP gates. The main points of interest of these algorithms is that they were able to improve accuracy by focusing on noise rather than reducing the amount of SWAP gates used. While noise aware, these algorithms still follow a tradition routing goal: insert SWAP gates to allow a circuit to be run on hardware. This differs from the approach of this paper, which is to insert SWAP gates not only to allow a circuit to connect qubits but by also avoiding poor links.

2.3 Qiskit's Routing Algorithms

Qiskit contains multiple different routing algorithms that can be selected during the compilation process. The different options are summarized below.

2.3.0.1 Basic Swap

"Basic Swap" is Qiskit's most basic form of routing. For Basic Swap, an initial mapping, coupling map, and circuit to be routed is needed. for each CNOT gate, the algorithm checks if the current if the current coupling map has an edge between the two qubits of the gate. If the current layout does connect the two qubits of that gate, the algorithm determines the shortest (least amount of swaps) path to route the gate parameter qubits to a valid pair.

This approach is very simple and does indeed solve the routing problem, but it is not ideal. Its only metric for where to add SWAP gates is the shortest path. It does not factor in how different paths could affect the circuit down the line. The other algorithms in Qiskit attempt to reconcile that. [7]

2.3.1 Sabre Swap

SABRE swap is a routing algorithm that attempts to solve the routing problem in multiple unique ways. First, by attempting to create an ideal initial mapping. The authors of SABRE postulate that an initial mapping plays an extremely important role in the performance of a circuit, because an ideal initial mapping can greatly reduce the amount or even eliminate the addition of SWAP in a circuit. They obtain their ideal mapping by first selecting a random mapping, then running their routing algorithm, and using the final mapping generated by their routing as their new initial mapping.

The routing algorithm itself is a heuristic search that separates the circuit into layers of data dependencies and progressively routes the layers together. It does this by maintaining a list of gates that can currently be executed on hardware; that is: gates whose data dependencies have been resolved but cannot run on hardware due to the mapping (denoted F), and all other gates. Once F has been filled, a cost function is run on each gate in F, with the goal of finding a SWAP that reduces the

distance between all other gates in F. This is repeated until all gates in F are able to be executed on hardware, where they are then added to the execute list and new gates are added back into F. [1]

2.3.2 Stochastic Swap

Stochastic Swap attempts to solve the qubit routing problem by using random permutations. It goes through the circuit layer through layer applying random permutations to attempt to find an ideal solution for that layer. [7]

2.3.3 Lookahead Swap

Lookahead swap is a routing algorithm that looks ahead in the execution to see which swaps have the biggest effect on the rest of the circuit. It does this by breaking the circuit up into chunks and comparing all possible swap gate possibilities. It ranks each possibility based on how it would affect subsequent gates in the circuit. It then chooses the best options and repeats this process until all gates are mapped. [8]

2.4 Errors and Noise

There are two main types of errors caused by noise: coherence errors and gate errors.

2.4.1 Coherence Errors

Coherence errors are caused by fragile qubits "decaying" and losing their state. The probability of a qubit decaying into another state increases with time. The amount of time associated with a delay of a quantum state $|1\rangle$ into a $|0\rangle$ is refered to as the T1 Coherence Time. Beyond simply decaying over time, a qubit can also fall into the $|0\rangle$ state from environmental factors. This is referred to as the T2 coherence time. [1]

2.4.2 Gate errors

Gate errors are errors that occur on the operation of a gate. They can occur on either two or one qubit gates. Gate errors are what makes deep circuits not viable as errors stack. For instance, if the gate error rate of a system was 20%, the total accuracy of the system after three gates would be $(1 - 0.2)\exp\{3\} = 0.512$. In a multi qubit system these errors occur on both single qubit gates and two qubit gates. This work focuses on two qubit gates, as the error rate varies for each link and can be used by the routing process.

2.4.3 IBM Noise Calibration

Given how important noise is on a quantum system, IBM regularly collects and publishes their data on the noise of each system. They publish this data using their IBMQ API, where users can incorporate the data into their circuits. Figure 2.6 shows a noise map generated by IBM for one of their 7 qubit architectures.

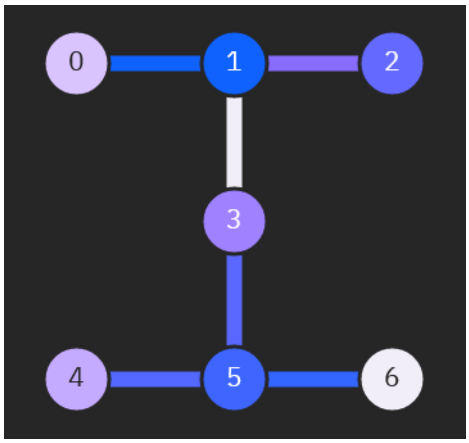


Figure 2.6: IBM generated noise map for their Jakarta system

The architecture shown in Figure 2.6 is an architecture that HERR is simulated against. In the generated graph each node represents the qubit and the link color between the qubits represents the error rate. The lighter color links indicate a lower error rate a darker color link represents a higher error rate. Figure 2.7 shows another

IBM generated noise map. This noise map is for their larger Guadalupe system.

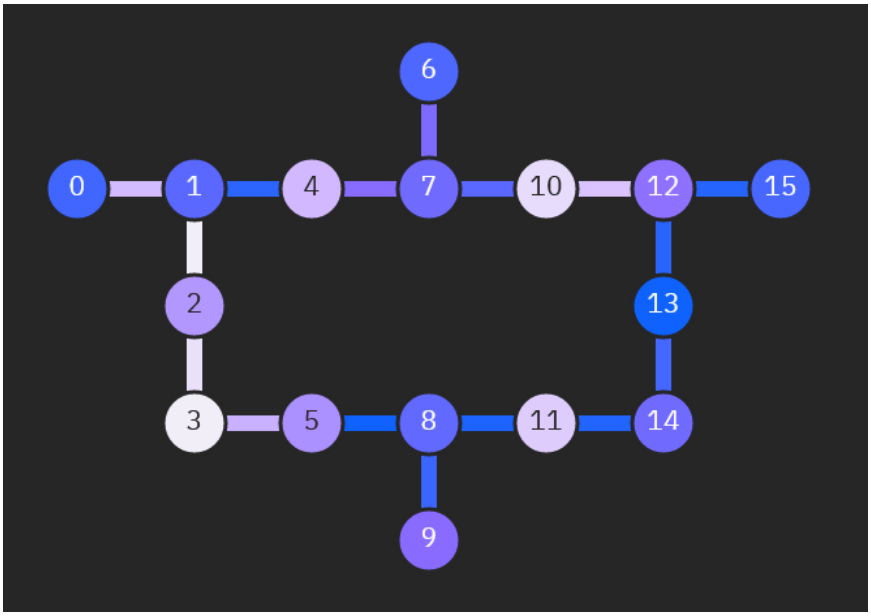


Figure 2.7: IBM generated noise map for their Guadalupe system

2.5 Existing Noise Adaptive Mapping and Routing

This is not the first work that attempts to address the routing problem through noise, and while our method is unique there are two other works that address the problem.

2.5.1 Noise adaptive routing

In this work, Murali et Al create a noise adaptive mapping and routing scheme that is able to acheive a 2.9x success rate of circuits over Qiskit. They first propose two initial mapping policies. The goal of these policies is to reduce the communication distance and therefore required SWAPs for a circuit. They then propose two main noise adaptive routers: one using satisfiability modulo theory (SMT) solvers, and one using heuristics.

SMT solvers are are programs that take in sets of constraints and attempt to produce an output that meets all constrainst. Murali et al designed their contraints

to attempt to produce a routed circuit that minimizing noise levels. They also create a heuristic routing method that does not use a solver. This method using their initial mapping algorithms to the initial mapping, then uses their "best path" routing algorithm. This "best path" routing algorithm works by assembling a noise map of all the qubits and uses Djikstra's algorithm with the noise of each link as the weight to compute the best path from each qubit to another. During routing their algorithm uses these precomputed values to determine the best SWAP path. [2]

2.5.2 Variation aware routing

Tannu and Qureshi tackle the mapping problem in a similar method as [2]. They develop what they call **Variation-Aware Qubit Allocation (VQA)** and **Variation-Aware Qubit Movement (VQM)**, where VQA is simply an initial mapping algorithms and VQM is a noise aware routing algorithm. For VQM, the cost, which is the total error rate, of each path during a routing operation is calculated and a path with the least cost is chosen. This approach is very similar to in [2]. VQA acts by ranking how frequently a qubit is used, and mapping the most frequently used qubits to qubits with the strongest links.

Chapter 3

High Error Rate Routing

3.1 Adding Swap Gates Can Increase Accuracy

Before diving into the intricacies of the Qiskit transpiler and how the High Error Rate Routing (HERR) algorithm works, it is important to justify the possible improvement using examples. Here we demonstrate how HERR algorithm can be used to increase the accuracy of two popular circuits: The Toffoli Gate, and the Quantum Fourier Transform (QFT).

3.1.1 Toffoli Gate

The Toffoli gate is ubiquitous in quantum computing. It can be thought of as doubly controlled CNOT gate, where there are two control qubits. Figure 3.1 shows the quantum circuit representation of a Tofolli gate. Both qubits must be 1 in order to flip the target qubit. The problem with Toffoli gates is they are not directly implemented in the hardware of quantum computers. Instead a Toffoli gate is broken down into the native gates of a given architecture (usually consisting of CNOT and

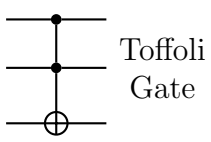


Figure 3.1: A Toffoli Gate

other single qubit gates). In the IBM superconducting architecture the Toffoli gate is decomposed as shown below.

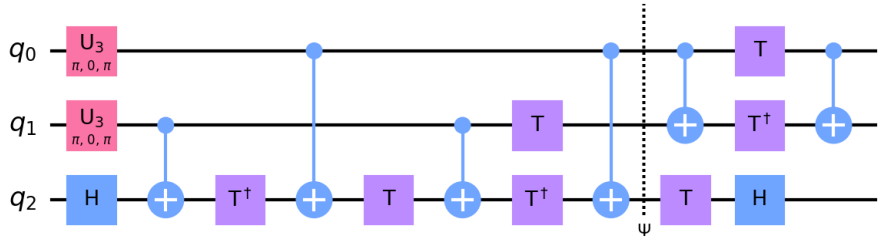


Figure 3.2: Decomposition of Toffoli Gate For an IBM Computer. Generated by Qiskit

Figure 3.2 shows that a single Toffoli gate is broken into 6 CNOT gates and a handful of single qubit gates. An important thing to note is that all three qubits must interact with each other using two CNOT gates. This can be a problem if the mapping or routing of a circuit includes a poor link between qubits. Take for instance a coupling map as shown below.

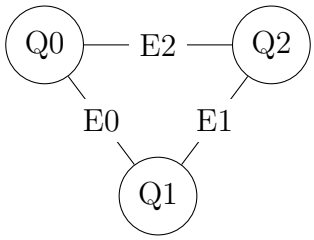


Figure 3.3: 3 Qubit Coupling Map

Suppose the logical qubits q_0 through q_2 from Figure 3.2 are mapped to $Q0$ through $Q2$ in Figure 3.3 such that $q_0 \rightarrow Q0, q_1 \rightarrow Q1, q_2 \rightarrow Q2$. Also assume the accuracy of Edges $E0$ through $E2$ are $\{0.90, 0.99, 0.99\}$. Note the two CNOT gates after ψ. They operate between $Q0$ and $Q1$, which corresponds to $E0$, which has a much lower accuracy than the other edges. In all current routing algorithms, this circuit would not be changed to add a swap since it is not needed. However, if a swap was introduced between q_1 and q_2, that means the CNOT gates would now be acting on $E2$ which has a much higher accuracy. The modified circuit is shown below.

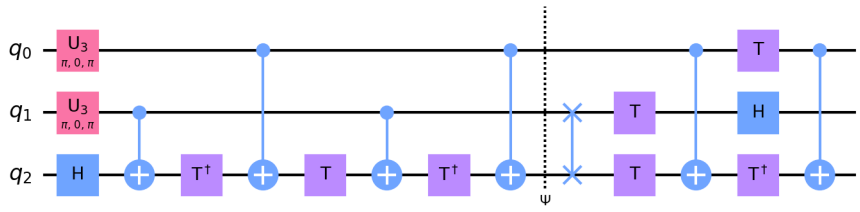


Figure 3.4: Figure 3.2 with an added swap gate at point ψ

Figure 3.4 shows the same circuit with a swap gate added at ψ. This allows the circuit to completely avoid the poor edge $E0$. This solution is not with its drawbacks however. A swap gate itself is decomposed into three CNOT gates, which increases the circuits depth and therefore could decrease the accuracy of the circuit. So the swap would only be worth it if the difference in accuracy between the two edges is large enough. For this specific example, a simulation was run in Qiskit to determine how it would affect the circuit's accuracy. Adding the swap gate to avoid the poor link, $E0$, increase the overall circuit's accuracy by 14%.

3.1.2 Quantum Fourier Transform

Like the Toffoli gate, the Quantum Fourier Transform (QFT) is a very important circuit in quantum computing. It is used in many famous algorithms, including Shor's algorithm. The general idea of the algorithm is to translate a number encoded in the computational basis onto the phase of qubits. The encoding can then be recovered using the inverse QFT. The general circuit for encoding and decoding a 3 bit number using the QFT is shown below.

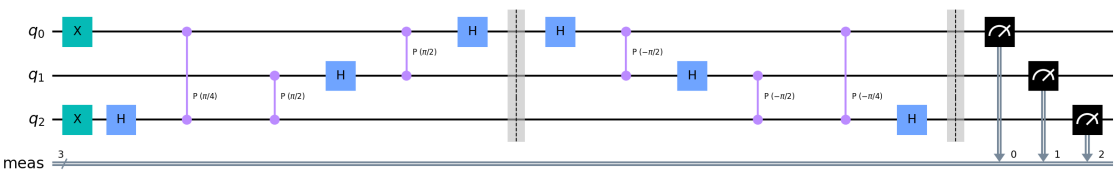


Figure 3.5: QFT Circuit that encodes then decodes $|3\rangle$

Figure 3.5 shows the basic three qubit QFT and phase estimation. Now assume the coupling map and edge accuracy rate are the same of the Toffoli gate example. The controlled rotation gates between q_0 and q_1 pose the same problem as the Toffoli gate, but exacerbated. Since each CNOT gate decomposes into two CNOT gates, there are four CNOT gates operated on this poor link. We could avoid this problem by adding extra SWAP gates that shift the links being operated on. The resulting circuit would look like the one below.

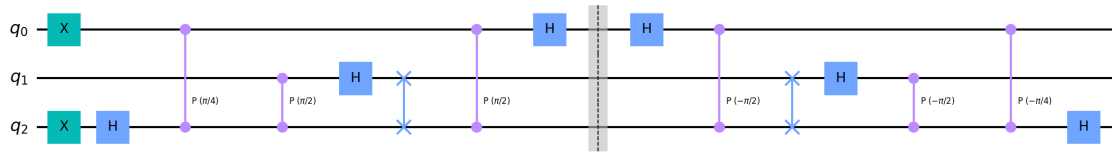


Figure 3.6: QFT Circuit that encodes then decodes $|3\rangle$ with additional swap gate for noise reduction

Figure 3.6 shows the two swap gates added that allow the CNOT gates to avoid the poor link that is q_0 and q_1. This change, simulated in Qiskit, lead to an 12% increase in accuracy of the circuit.

The last two sections have made the case that introducing un-needed swap gates can create improvements in some circuits, but in these examples swap gates were manually added. In the next sections we introduce an algorithm that automatically analyzes a circuit and inserts swap gates in an attempt to increase accuracy.

3.2 HERR: High Error Rate Routing

The idea presented in the previous section is the basis for the HERR (High Error Rate Routing) qubit routing algorithm. This algorithm was implemented in Qiskit with the intention of integrating with the Qiskit transpiler.

Like all other Qiskit routing algorithms, HERR mainly operates on the DAG

(Directed Acyclic Graph) that represents a circuit. The dag splits each circuit into layers of independent operations. HERR iterates through each layer and through each two qubit gate of that layer to look for opportunities for adding a swap gate. In order for HERR to determine if an extra swap is appropriate, it must consult the noise map. The noise map is identical to the coupling map, but with weighted edges. The weight of each edge represents the error rate between the two qubits. A noise map can be generated from data IBM publishes about their quantum systems. IBM routinley calibrates and reports error rates for each link [1]. An example of a noise map is shown below.

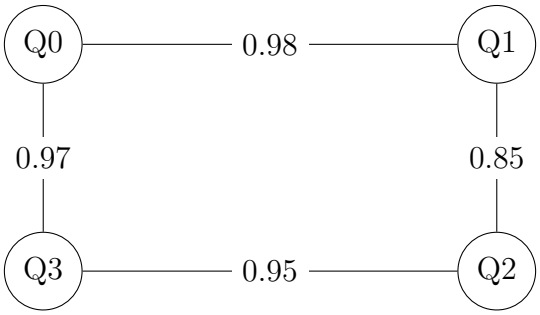
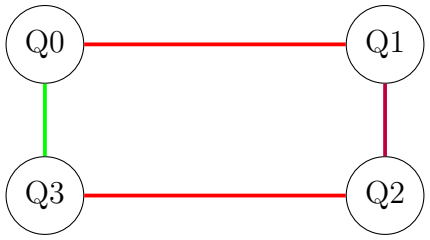
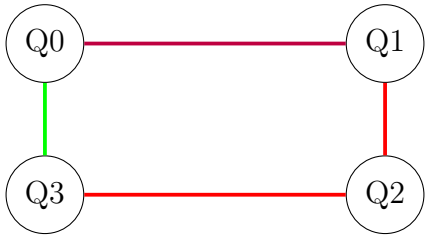


Figure 3.7: Example noise graph For a 4 Qubit system. The values on each edge represent the error rate for a CNOT gate operating on the two qubits

Figure 3.7 is an example noise graph for a four qubit system. For each CNOT gate in the circuit, the algorithm calculates the predicted error rate if the operation was run as is, and then calculates the predicted error rate of all possible alternatives. It searches the coupling map and noise graph for alternatives using a breadth first search. It limits it search after a predetermined distance from the original link. The distance in this instance refers to the maximum amount of swaps needed by each argument qubit to route to the new link. This is also known as the L1 norm distance also known as Manhattan distance. An example is shown below.



(a) Edge q1, q2, shown in purple, has a distance of 1 from edge q0, q3, shown in green, since one swap gate is needed per qubit. The path for each qubit is shown in red.



(b) Edge q0, q3, shown in green, has a distance of 2 from edge q0, q1, shown in purple, since two swap gates are needed to route q3 to q1. The path for q3 to q1 is shown in red.

Figure 3.8: An example of a distance of one and two in the context of qubit links

Figure 3.8 shows how distance is computed for HERR. Figure 3.8a shows what a distance of one between two edges looks like, while Figure 3.8b shows a distance of two. The HERR algorithm has a configurable depth.

Each edge within the configured distance is considered, and a cost function is run for each edge to determine if it is better than the old edge. If a better edge is found, swap gates are inserted such that the shortest path is taken between the edges.

3.2.1 Path Determination

In order for HERR to determine if a given edge is better, HERR must decide on a path to that edge. For instance, suppose the coupling map shown in Figure 3.9 is being used, an operation is being performed on E0, and HERR is attempting to determine if E3 would be a better edge. Moving from E0 to E3 means that the arguments of the CNOT gates would transfer from $\{Q0, Q1\} \rightarrow \{Q3, Q5\}$. As a

reminder, Qn represents a physical qubit while qn represents a logical qubit. There are two options for how this transformation could be done. Suppose the logical to physical qubit mapping was $\{q0 \rightarrow Q0, q1 \rightarrow Q1\}$. HERR could either update the mapping to $\{q0 \rightarrow Q3, q1 \rightarrow Q5\}$ or $\{q0 \rightarrow Q5, q1 \rightarrow Q3\}$. While these options may seem equivalent, the path they take isn't and can drastically affect the accuracy. Suppose HERR chose the transformation as $\{q0 \rightarrow Q5, q1 \rightarrow Q3\}$. A SWAP gate would be inserted across E2 to map q1 to Q3. But to map Q0 to Q5, SWAP gates would need to be inserted at E0, E2, and E3. But since a swap gate was inserted at E2, the SWAP for mapping q1 to Q3 was undone, and a SWAP on E2 must be performed again. This means a total of 5 SWAP gates were needed. Suppose to obtain the same mapping q0 was mapped to Q5 first. SWAP gates would be inserted at E0, E2, and E3. But since a SWAP was inserted at E0, this mapped q1 to Q0. So to map q1 to Q3, SWAP gates would be required at E0 and E2. This would make the total number of SWAP gates 5, which is the same as mapping q1 to Q3 first.

What if the mapping was instead $\{q0 \rightarrow Q3, q1 \rightarrow Q5\}$? Two SWAP gates would be needed at E2 and E3 to map q1 to Q5, and two SWAP gates would be needed to map q0 to Q3. This totals 4 SWAP gates instead of 5 of the other mapping. This greatly reduces the error accumulation, since each SWAP gate decomposes into three CNOT gates.

This idea can be generalized such that when HERR contemplates a path, it always tries to choose the shortest route. In this above example this is that path that takes 4 SWAP gates. Normally, finding the shortest path if trivial, but for HERR it is slightly harder because we are not just finding the path for one qubit, as is done during normal routing. Since we are finding the path for both qubits, we need to make sure we avoid the "doubling back" that can happen if the path for a SWAP goes through the other qubit. This "douling back" bug was the effect described in the above example when it took 5 SWAPs.

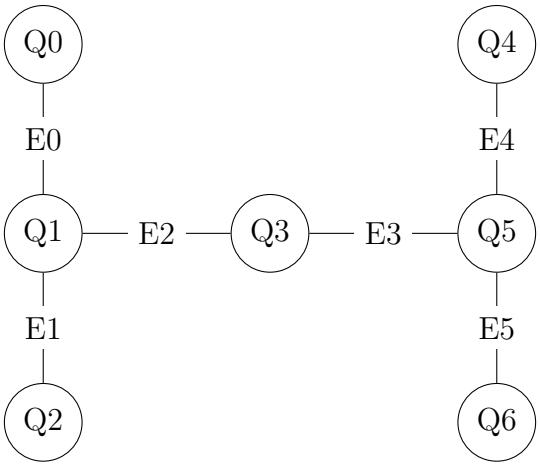
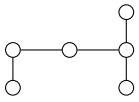


Figure 3.9: Example Coupling Map



(a) Coupling Map when attempting to route q1 to Q5

(b) Coupling Map when attempting to route q0 to Q3

Figure 3.10: Examples of HERR generation subgraphs for routing

HERR avoids this by determining the shortest path using a sub-graph that excludes the other qubit. For example, during the routing process for the example above, when attempting to route q1 to Q5, HERR would generate a subgraph that excludes q0. An example is shown below.

Figure 3.10a shows the subgraph HERR would generate when attempting to route Q1 to Q5. Notice how Q0 is not present. This is because the generated subgraph excludes the other qubit so that the "doubling back" does not occur. For this instance, this is not a problem, as there is a clear route between them. Figure 3.10b shows an instance where this would be a problem. Here, we are viewing the subgraph generated by HERR when trying to route Q0 to Q3, excluding Q1 which is the other argument. This subgraph is just Q0, and does not contain the target Q3. This raises a flag in HERR, and is how HERR realises that Q0 must not be routed to Q3 first. When performing path finding HERR generates a subgraph a tries to find a valid path for all options. While this is not very time efficient, it can yield greater efficiency.

3.2.2 Cost function

The basis of HERR is determining if the cost of swapping to a new link is less than the cost of keeping the current layout. To facilitate that decision, HERR uses a cost function to evaluate the effectiveness of a link. The cost function predicts the accuracy associated with a new link by calculating the error associated by all the SWAP gates needed to route logical qubit to a new physical qubit.

Suppose a circuit were run on the coupling map shown in Fig. 3.11, and the circuit calls for a CNOT gate that operates on Q0 and Q3 (shown in green). As part of the noise aware routing process, HERR considers if E0, between Q0 and Q1 (shown in purple), is a better link. The path in this case would be $Q3 \rightarrow Q2 \rightarrow Q3$ and requires SWAP gates on edges 1 and 2. The cost function calculates the predicted accuracy as the product of the accuracy of E0 and the accuracy of inserted SWAP gates on E2 and E1. Since a SWAP gates decompose into three CNOT gates, the total accuracy of a SWAP gate is the cube of the accuracy of the link it operates on. This calculation is shown in Eq. 3.1 where $P(A)$ is the predicted accuracy of moving to a different link. In this equation a_E is the accuracy of the target link. Each product term refers to the error accumulated by the path each qubit of the argument takes, ie $\prod_i^j a_{0,i}^3$ to the error accumulated by qubit 0 traveling from initial placement i to final placement j. The path as a list of edge accuracies. For instance, suppose a path on the noise map in Figure 3.7 was from Q0 to Q2, going through Q3. The path would be $0.97, 0.95$. The term $a_{0,i}$ is cubed because each SWAP gate decomposes into 3 CNOT gates operating on the same link. Likewise, $\prod_{i=0}^n a_{1,i}^3$ refers to the path qubit 1 takes. The reason the product is used is because to determine the final accuracy of a path each step of the path must be multiplied together. If one qubit is not moving, the corresponding product term is 1.

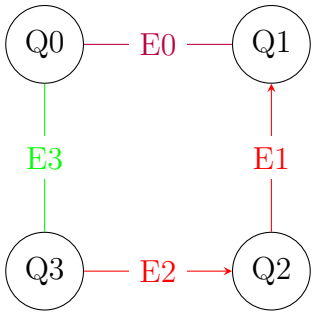


Figure 3.11: Path taken when swapping to Edge 0 from Edge 2

$$P(A) = a_E \prod_{i}^{j} a_{0,i}^{3} \prod_{i}^{j} a_{1,i}^{3} \tag{3.1}$$

HERR compares the predicted accuracy calculated by the cost function and compares it against the accuracy of the original link the gate is supposed to act on. If the predicted accuracy is better, it will swap to the new link.

Recall the example shown in Figure 3.8b. We are trying to move a qubit from Q3 to Q1 through Q2. If the noise map of the system is that of Figure 3.7, the accuracy from Q3 to Q2 is 0.95 and the accuracy from Q2 to Q1 is 0.85. Since the other qubit is not moving, only one term in Eq. 3.1 is used. The result of plugging the accuracy values into Eq. 3.1 is shown in Eq. 3.2.

$$P(A) = 0.98 * 0.95^3 * 0.85^3 = 0.516 \tag{3.2}$$

The result of Equation 3.2 shows that this path would yield a total accuracy of 0.516, which is much less than the original link accuracy of 0.97. Therefore this path would not be taken.

3.3 Standard Routing

The above section describe how HERR operates as a noise aware routing. The key point is that during the routing process HERR looks for opportunities to add SWAPs that can increase accuracy. This function of HERR does not fully solve the routing problem, as it does not describe how HERR operates when it encounters a gate whos argument qubits are not connected on the coupling map. This is the standard definition of the routing problem, which is what the Qiskit's routing algorithms solve.

During standard routing operations HERR acts just like Qiskit's basic swap. It only adds SWAP gates when when two qubits need to be routed together, and in that case it uses the shortest path. This is in contrast with qiskits other routing algorithms that attempt to minimize the number of SWAP gates added. The only rationale behind this choice was time. Basic Swap is trivial to implement, and adding HERR on top of it is an easy way to show the benefits of HERR. HERR is flexible enough that it could be modified to perform more complex routing procedures for normal routing, such as Qiskit's SABRE or stochastic SWAP. This idea is expanded upon later.

Another important design consideration for HERR was its initial mapping. It is known that initial mapping is an important choice, as a good initial mapping can reduce the amount of SWAP gates needed to route a circuit. For HERR, we choose a trivial initial mapping. This is a mapping such that each logical qubit is mapped to its corresponding physical qubit, Ie, $\{q0 \rightarrow Q0, ..., qn \rightarrow QN\}$.

3.4 Psuedocode

The execution logic of HERR is not very complex. Essentially, the circuit is converted into its DAG representation, and then split into layers. HERR then iterates over each gate in each layer, and attempts to find a mapping. HERR runs the cost function for

inserting SWAP gates to each other qubit to determine if the current gate would have a higher accuracy on a different link. If so, HERR will add SWAP gates to perform the routing. This process is outline in Algorithm 1.

Algorithm 1 Finding a better link

```
procedure FindBetterLink(g)# Finds better link for gate g
    E_best ← E(g)
    a ← C(g)
    for all Edge e in Coupling Map do
        if C(e) > a then
            E_best ← e
            a ← C(e)
    return E_best
```

Beyond finding a better link, HERR also must use the resulting link from Algorithm 1 to modify the circuit. This procedure is outline in Algorithm 2.

Algorithm 2 Finding a better link

```
CircDag ← circuit.generateDag()
for all gate g in CircDag do
    if couplingMap.isConnected(g) then
        currLink ← couplingMap.GetEdge(g)
        possibleLink ← FindBetterLink(currLink)
        if possibleLink is not None then
            CircDag ← insertSwaps(currLink, possibleLink)
    else
        performBasicSwap(CircDag, g)
```

Algorithm 2 shows how HERR operates on a high level. For each gate in a circuit, it first sees if the arguments are connected. If they are, it will attempt to find a better link using Algorithm 1. If no link is found, it will do nothing. If it finds a better link it will insert the SWAPs needed to switch to that link. In the case the arguments of the gate are not connected on the coupling map, it will perform the BasicSwap operation, which is the same as Qiskit's BasicSwap algorithm.

Chapter 4

Experimental Results

4.1 Benchmarks and Metrics

4.1.1 Benchmarks

To test HERR, three popular benchmarks were chosen: The Quantum Fourier Transform (QFT) [3], the Bernstein-Vazirani (BV) algorithm [9], and a Toffoli gate [3]. These benchmarks were chosen primarily because of their use as benchmarks by other mapping algorithms [1][2], as well as their relevance in the field. A benefit and the QFT and BV algorithms is also their ability to scale to different hardware sizes, which allows to analyze how larger qubit circuits run under HERR. The QFT and Toffoli gate were introduced in the previous chapter, and here we briefly introduce the Bernstein-Vazirani algorithm

4.1.1.1 Bernstein-Vazirani Algorithm

The Bernstein-Vazirani algorithm is an extension of the Deutch Joza algorithm, and is used to find a binary string encoded in a function [10]. It is an oracle based algorithm, where the binary string is encoded into the oracle. Figure 4.1 shows an example of the BV algorithm for a binary string of "1101".

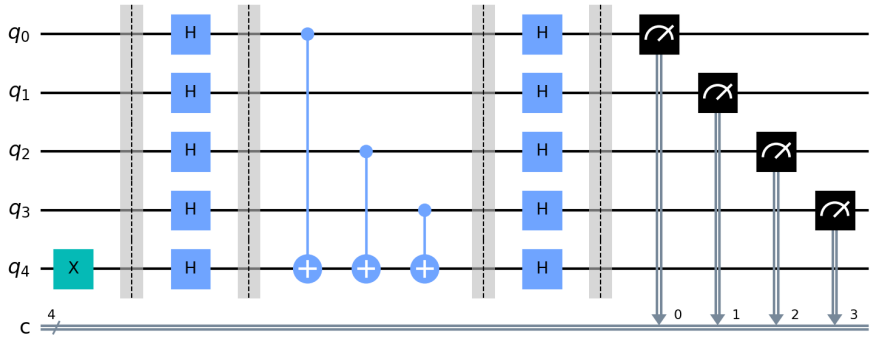


Figure 4.1: Bernstein-Vazirani Algorithm for a binary string of 1101. Generated by Qiskit

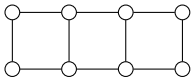
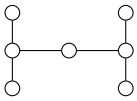
The BV algorithm is useful as a benchmark because it is simple to implement, and the depth is shallow enough that huge amounts of error do not accumulate. It is also an oracle based algorithm, so it gives a glance into how HERR could perform on other oracle based algorithms such as the famous Grover's algorithm.

4.1.1.2 Coupling Maps

Beyond algorithms, HERR is evaluated on three different coupling maps: a 4 qubit grid, an 8 qubit grid, and the 7 qubit architecture of a IBM Falcon r5.11H processor. Testing on the IBM architecture allows us to see how HERR performs on a real architecture, as well as a coupling map with lower connectivity between the qubits. The pairing of benchmarks to coupling maps are shown in Table 4.1

In addition, this paper only tests up to 8 qubit circuits. This is because circuits with more qubits usually have a greater depth, such as the QFT. On any circuits larger than 8 qubits, the increased depth causes so much noise accumulation that results from the circuit are unusable. This is a problem that was encountered by Tannu and Qureshi in their attempt at a noise adaptive mapping scheme. They introduced a metric to account for it called mean time before failure [1], which can be used to benchmark circuit with a greater depth, but this paper does not use this benchmark.

Table 4.1: Benchmarks and coupling maps used to test HERR

Benchmarks	Coupling Map
1. QFT4 2. BV4 3. Toffoli Gate	
1. QFT8 2. BV8	
1. QFT7 2. BV7	

Another important note to make about coupling maps is our choice to only test circuits that have a similar qubit size to the coupling maps they are run on. For example, we only test smaller circuits (BV4, QFT4, Toffoli) on a 4 qubit coupling map, and tested their larger versions on larger coupling maps (BV8 and QFT8 on the 8 qubit coupling map). we wanted to see how the routing algorithm performed when there is a limited choice of available links and qubits. There is a valid case for running small circuit, like the QFT4, on larger coupling maps. This could be beneficial, as it would give HERR more potential to find better links. That said, we believe this is out of scope for this thesis, and is better suited for future work.

4.1.2 Metrics

Three metrics are used: accuracy, number of CNOT gates, and routing execution time. These metrics were chosen because we believe they are the most relevant metrics for choosing a routing algorithm, and they are explained below.

The accuracy of a benchmark is defined as the percentage of trials where the output of a circuit was correct. This metric is perhaps the most important, as accuracy is the biggest hurdle in noisy quantum systems. A routing algorithm with a higher accuracy will almost always be preferred because it produces a more useful result. While seemingly an obvious metric, other routing algorithms do not actually benchmark themselves on accuracy, instead measure the number of added SWAP gates [6]. This is because the number of gates has been used as a proxy for accuracy, since when noise is not accounted for, gate depth is the main contributor to error. This is why we also use number of CNOT gates added as a metric.

Using the amount of CNOT gates added as a metric allows us to compare the efficiency of routing algorithms. While we have previously said the amount of CNOT gates is the not best measure of a routing algorithm as noise levels can make it beneficial to use more CNOT gates than needed, it is still an important metric as it allows us to compare how HERR, an algorithm that does not focus on efficiency, but rather seeking strong links, compares to existing algorithms that do focus on efficiency. Higher rates of CNOT gates grow the complexity of the circuit, but may come with better or worse accuracy results.

The final metric we focus on is routing execution time. This is important because routing is a heuristic NP hard problem and while an algorithm may find an good solution, if the routing time does not scale well with circuit size the algorithm could be unusable for large circuits.

4.2 Test Environment

The results were extracted through simulation on Qiskit Aer. Qiskit Aer allows for various types of noise sources to be simulated for a circuit. This allows us to test how HERR performs under various noise environments that hardware would not easily allow.

The noise model generates a random noise value for each link in the coupling map. Each noise source is a depolarizing error in Qiskit, since it can act on a link rather than just one qubit. The transpiled circuit for each routing method is used for the simulation process. Each circuit is circuit is simulated 1024 times for each randomly generated noise mapping.

We simulated two noise ranges: 1-10% and 1-20%. This two values were chosen based off observations from IBM own calibration data. It was observed that error rates ranged from 1-20% system wide, but the large majority of the links were within 1-10%. Therefore 1-10% was chosen to represent a more typical system, while 1-20% was chosen to represent a noisier, but still realistic system.

4.3 Results

4.3.1 Routing Time

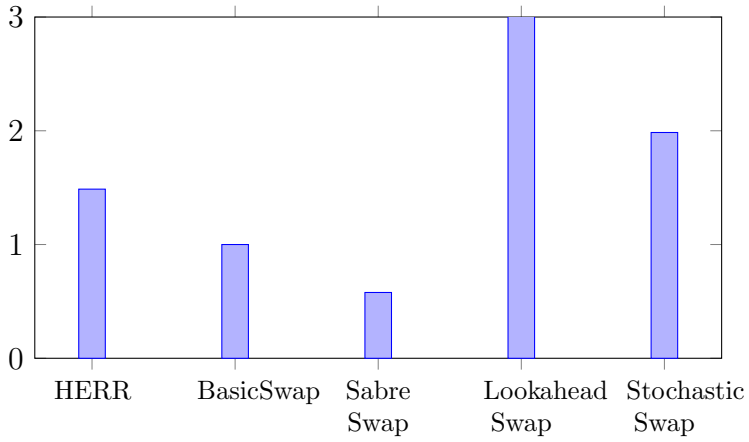


Figure 4.2: Average routing time by routing algorithm normalized against BasicSwap. Note that the value for Lookahead (77x) does not fit in the y-axis to allow clarity for the other benchmarks.

The time that it takes to complete the routing process is an important metric to take into account, specially as circuits grow in size in the future. Figure 4.2 shows the average normalized time taken for each routing method to perform routing, against

Table 4.2: Normalized routing time for all benchmarks, best cases highlighted%

Bench mark	HERR	Basic	Sabre	Lookahead	Stochastic
QFT4	1.62	1.00	**0.42**	44.87	1.75
QFT7	1.25	1.00	**0.57**	82.95	1.84
QFT8	2.00	1.00	**0.67**	107.31	2.57
BV4	**0.35**	1.00	0.50	3.80	0.93
BV8	1.03	1.00	**0.32**	27.09	1.17
Toffoli	0.82	1.00	**0.51**	18.09	0.98

Qiskit's BasicSwap routing method. Table 4.2 shows the details of each experiment. The first thing to notice is the very large difference on the time that takes to calculate the best routing for the Lookahead routing algorithm. Figure 4.2 does not include the full column for averaging over 77x that of BasicSwap. Adding a swap gate triggers changes in the future routing problem, and Lookahead looks into all the possible future issues, hence the extreme cost of its routing, which will be prohibitive in future, larger circuit implementations. When it comes to routing time, HERR takes about 39% more time than BasicSwap on average, which is expected for performing the BasicSwap and high error routing together. HERR actually performed better than BasicSwap for a couple of cases (BV4 and Toffoli). The reason behind this is that the poor link avoidance helped the rest of the routing for those problems. Sabre is, on the other hand, much more efficient from a routing point of view, since it focuses only on a group of gates at a time. The random permutation approach of Stochastic is not particularly efficient in terms of routing time resulting on an average 2x routing time, with some minimal improvements for some benchmarks.

As expected, larger circuits (QFT8 for instance) require more time than the others. The most efficient benchmark from the routing time perspective for all routing algorithms is BV4, while QFT8 was overall the worst performing one. This is because of the depth of and number of qubits in a circuit. The BV algorithm is a very shallow circuit, as it only consists of 2 layers of Hadamard gates and a simple oracle. This is in stark contrast to the QFT which is a very deep circuit. The QFT has many

layers as can be seen in Figure 3.5. A greater depth requires more processing as each layer needs to be routed. This is especially true since QFT has more two qubit gates, which means more routing needs to be done to ensure connectivity. This can be seen in Table 4.2. Comparing BV4 to QFT4, QFT4 has a longer routing time than for all except SABRE swap. This is because the depth of the QFT. Table 4.3 gives context to how many more CNOT gates QFT has than BV. It shows the total number of CNOT gates in a circuit before routing is performed. QFT8 contains over 22 times as many CNOT gates as BV, so it is no surprise that it took much longer to route.

The effect of depth on routing time is also important on how circuits scale. Figure 4.3 shows how an increase in circuit size affects the routing time. It shows that the jump from 4 to 8 qubits increased the routing time by a much larger degree for the QFT rather than BV. This makes sense since the BV algorithm gains much less depth when jumping to 8 qubits than does the QTF.

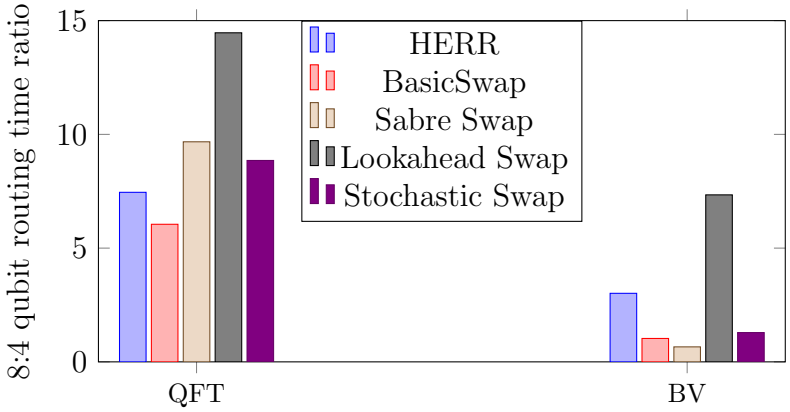


Figure 4.3: Ratio of routing times for BV and QFT for their 8 and 4 qubit implementations

Figure 4.3 also shows that HERR scales very well with an increased depth, especially for the QFT. For the QFT, only BasicSwap saw a smaller ratio of routing times. One possible reason for this is that since the QFT gains much more depth at higher qubit levels, other algorithms have to spend much more time attempting to find an efficient routing. This is in contrast to HERR which performs a more trivial routing that does not try to find the most efficient routing, but rather avoids noisy

Table 4.3: Number of CNOT gates used for each benchmark, before routing

Bench mark	# CNOT Gates
QFT4	36
QFT7	102
QFT8	136
BV4	2
BV8	6
Toffoli	6

links. The figure also shows that HERR scaled poorly for the BV algorithm. One possible reason for this is that the increase in depth going from BV4 to BV8 is much smaller than the QFT, and Qiskit's routing algorithms spend much less time finding an ideal routing, while HERR spends more of its time trying to find better links

These observations could be generalized as follows: HERR routing time scales better for deeper circuits, as its algorithm for avoiding noisy links scales much better than spending time trying to find an efficient routing.

4.3.2 Added CNOT Gates

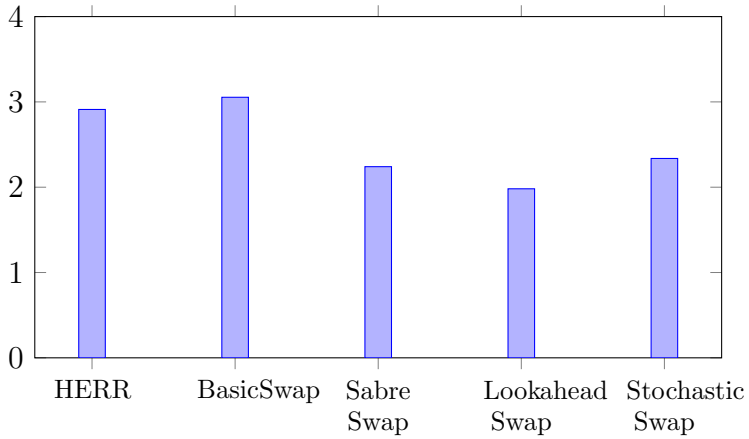


Figure 4.4: Average ratio of CNOT gates for each routing algorithm (after routing:before routing)

Figure 4.4 shows the average ratio of CNOT gates after routing to before routing, with details in Table 4.4. This is essentially a measure of how efficient a routing algorithm is. A more efficient algorithm has a lower ratio of added CNOT gates, since

Table 4.4: Ratio of gates added after routing for all benchmarks, best cases highlighted%

Bench mark	HERR	Basic	Sabre	Lookahead	Stochastic
QFT4	1.22	1.25	1.25	**1.00**	1.08
QFT7	3.17	3.26	1.91	**1.88**	2.47
QFT8	2.54	2.61	1.88	**1.71**	2.37
BV4	3.84	4.00	**2.50**	**2.50**	**2.50**
BV8	4.64	5.20	3.40	2.80	3.60
Toffoli	2.07	**2.00**	2.50	**2.00**	**2.00**

a lower ratio means less CNOT gates were needed to complete routing. Efficiency is an important metric because each CNOT gate adds additional error, so less is generally better. Basic Swap, for instance, is the least efficient from this point of view, multiplying the number of CNOT gates over 3x on average for the tested benchmarks, with HERR following closely. Basic Swap does not attempt to minimize the number of added SWAP gates, but simply to create the necessary links among qubits that need to interact. Therefore, is not a surprise that it perform worse than other algorithms that attempt to minimize these gates —Sabre, Stochastic and Lookahead. It is also not a surprise that HERR is very close to BasicSwap, as when not avoiding poor links HERR performs BasicSwap. The goal of HERR is not to minimize the number of gates, but to avoid damaged links, so it is a pleasant surprise to see that it does result in a slightly lower number of CNOT gates than Basic. Lookahead is the most efficient, since it considers all future routing problems and its goal is to minimize the number of SWAP gates (3 CNOTs each).

However, as we have seen this is at a very high cost in routing time. SABRE swap is right behind Lookahead as the next most efficient algorithm. This is due to SABRE's unique routing method that slices a circuit into layers and attempts to route layers together with help of a cost function. Stochastic Swap, which uses random permutation to try a find an efficient mapping, is very close but just behind SABRE swap.

Out of all the benchmarks, BV results in the highest addition in terms of gates.

This is due to the highly connected structure of this application circuit. The oracle of a BV circuit can have a large amount of CNOT gates that all act on an ancilla qubit, which means a large amount of SWAP gates would need to be added to make sure all the oracle's CNOT gates can be executed. This is shown in Figure 4.1 qubits 0, 2, and 3 all act on the ancilla. This heavily loads that single ancilla qubit. QFT on the other hand spears out its two qubit gates more. It instead loads its qubit so each qubit pair interacts twice through out the circuit. This means theres more of a chance that mapping is valid for a gate and no additional SWAPs are needed.

One interesting observatino from Table 4.4 is that for QFT4, lookahead seems like it did not add any CNOT gates. This seems impossible at first, as the QFT4 cannot be run on the square coupling map without some SWAP gates being added. Upon further inspection it seems that lookahead was able to use some of the SWAP gate built into the QFT algorithm as part of the routing process, and thus looks like it added no gates. This is interesting, and shows that the increased time lookahead takes can yield large benefits.

4.3.3 Accuracy

Table 4.5: Accuracy (%) of all benchmarks with an error rate of 1-10%, best cases highlighted

Bench mark	HERR	Basic	Sabre	Lookahead	Stochastic
QFT4	24.6	20.5	**37.4**	36.7	35
QFT7	1.8	1.6	4.7	**4.9**	2.5
QFT8	1.7	0.9	2.1	**2.3**	1.3
BV4	84.2	80.5	**86.3**	82.7	74.6
BV8	50.9	45.3	**62.4**	61.8	57.5
Toffoli	**80.1**	77.8	68.4	73.5	72.7

Figure 4.5 and Tables 4.5 and 4.6 show the average accuracy and detailed accuracy for each benchmark for two noise levels, 10% and 20%. On average, HERR does not seem to significantly improve on other algorithms besides Basic. Unfortunately, the

Table 4.6: Accuracy (%) of all benchmarks with an error rate of 1-20%, best cases highlighted

Benchmark	HERR	Basic	Sabre	Lookahead	Stochastic
QFT4	12.5	10.5	19	**18.5**	14.1
QFT7	0.9	0.9	**1.9**	1.4	1
QFT8	0.4	0.4	0.5	**0.6**	0.4
BV4	**71.4**	65	59.6	70.1	56.7
BV8	46.4	30.3	**60.8**	53.3	37.6
Toffoli	**65**	61	46.6	54.8	57.5

QFT benchmarks perform terribly from an accuracy point of view for the larger sizes (7 and 8). This is because QFT7 and 8 are so deep that the accumulated error starts to have large effects and result in very poor accuracy. This is even more prevalent at even higher qubit sizes. QFT16 was attempted, but the error accumulation was so large the results were useless. This behavior was also exhibited in [1], where the authors developed a new metric, mean time to failure that measures how long a circuit executes before it fails. We determined that metric was out of scope for this thesis.

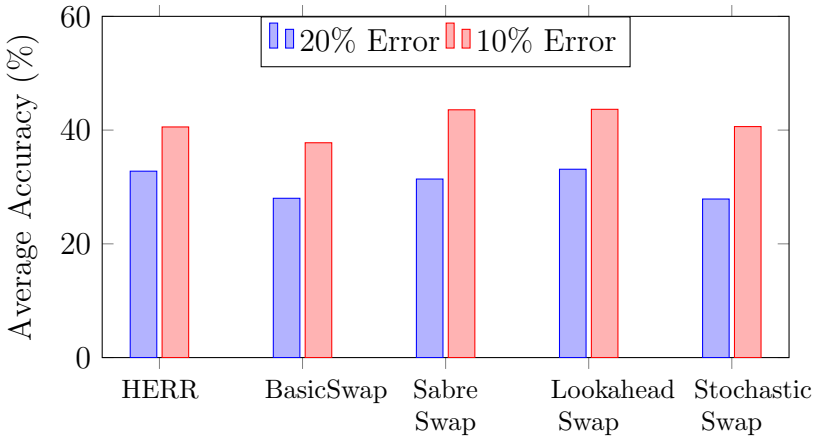


Figure 4.5: Average Accuracy for each routing algorithm with two noise ranges

Despite that, for the 20% noise average results, HERR actually surpasses all of them on average except for Lookahead, which performs equally (33% average accuracy for both vs. 31% for Sabre and 28% for Basic and Stochastic), with an enormous difference in routing time as shown in Figure 4.2. This is thanks to the good behavior

for BV4 and Toffoli, and the following not that far behind for other cases. Since HERR tries to improve upon routing on system with high error rates on specific links, it does perform better for the 20% noise case than the 10% noise rate case. Figure 4.6 shows the accuracy results for the 1-10% range for all benchmarks, and Figure 4.7 shows the same for the 1-20% range.

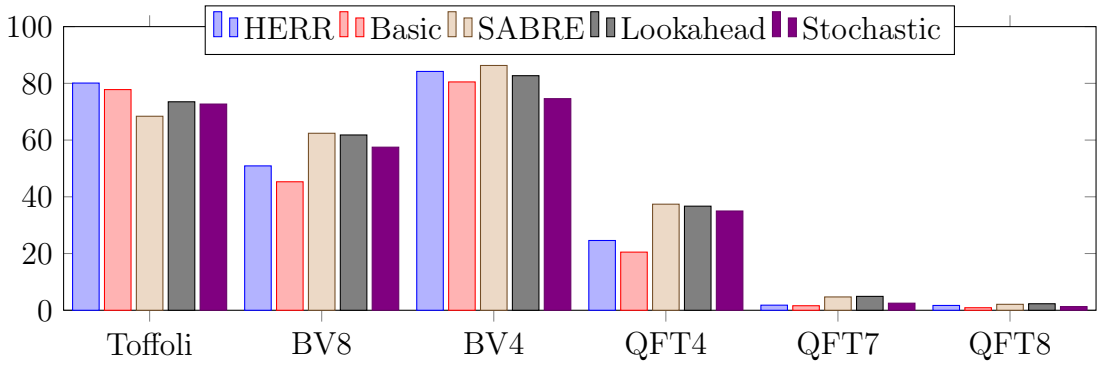


Figure 4.6: Accuracy of each benchmark when simulated with a noise range of 1 to 10%

Looking at the results for each application, for BV4 and Toffoli, HERR performs significantly better than all of the others. Putting Lookahead aside for its high routing time, it is interesting to see that for the BV4 case, Basic is the second in accuracy (71% and 65% for HERR and Basic, as opposed to 59% and 56% for Sabre and Stochastic). This indicates that *minimizing the number of CNOTs is not necessarily always a good indicator of accuracy*, specially when high error links are part of the qubit map. Also, HERR performed better than all others for both the 10% and 20% noise levels for these two benchmarks in particular (BV4 and Toffoli). The best performing routing algorithm seems to be highly dependent on a number of aspects, including the circuit application being targeted.

Previous works have made the assumption that fewer added CNOT gates yields a more accurate circuit. Figure 4.8 shows a plot of the ratio of added gates vs the accuracy of a benchmark. In this figure each point is a routing algorithms accuracy. This figure shows that all benchmarks besides BV4 have a slight downwards slope. This means that as the ratio of added gates gets larger (representing a more inefficient

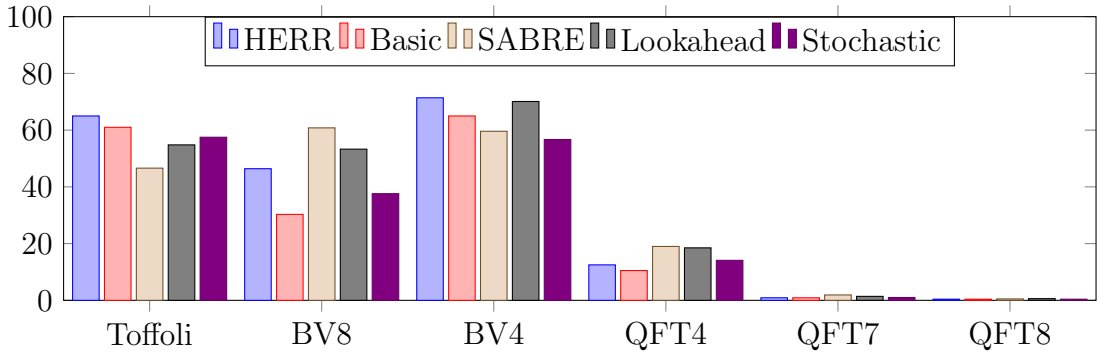


Figure 4.7: Accuracy of each benchmark when simulated with a noise range of 1 to 20%

algorithm) the accuracy is lower. This is expected. The one exception is BV4, which seems to be a flat line. Upon closer inspection, the two BV4 points around the gate ratio of 4 are BasicSwap and HERR. This is interesting, as it shows that while these routing algorithms were less efficient they performed just as well if not better. For HERR this makes sense, as the noise aware routing could have added extra SWAP gates to avoid a noisy link. It is less clear BasicSwap performed as it did.

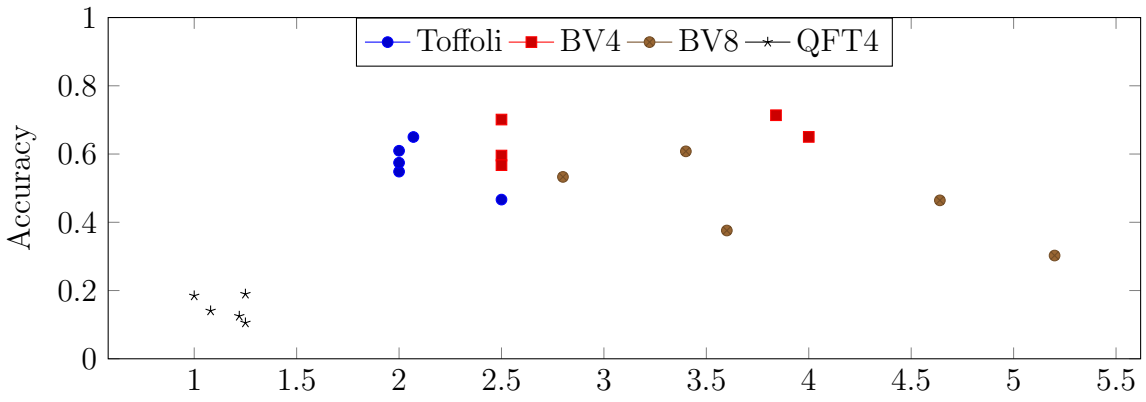


Figure 4.8: Accuracy plotted against ratio of added CNOT gates for selected benchmarks

Figure 4.9 shows the same information as Figure 4.8 but for QFT7 and QFT8. QFT7 and 8 are shown separately for clarity as their accuracy is so much lower than the rest of the routing algorithms.

Another metric that can be compared to accuracy is compilation time. Table 4.7 shows the ratio of accuracy to normalized compilation time. Normalized compilation

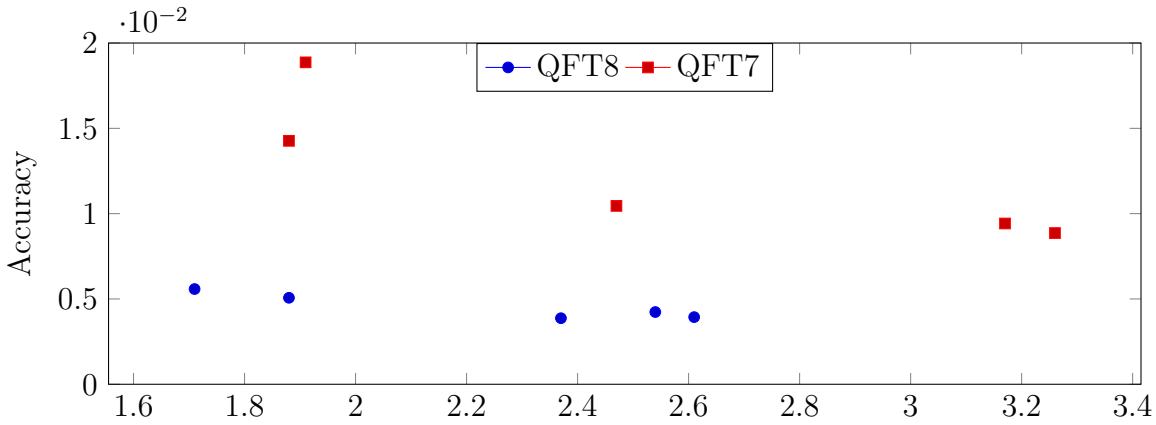


Figure 4.9: Accuracy plotted against ratio of added CNOT gates for QFT7 and 8

time is ratio of a routing algorithms compilation time to Basic Swap's compilation time, which can be seen in Table 4.2. The ratio of accuracy to routing time metric shows how time efficient a routing algorithm is. The higher the number the more time efficient the routing algorithm is for getting an accurate result. Figures 4.10 and 4.11 show a graph representing these values. These figures show that by far SABRE is the best algorithm for all except BV4, where HERR is the best. The plots also show that HERR is relatively time efficient for all benchmarks, but is very time efficient for BV4. HERR performed so well at BV 4 because a combination of HERR routing very quickly, as shown in Table 4.2, and having a superb accuracy as shown by Table 4.6. This excellent efficiency is only present for BV4, as for BV8 it is more in line with the rest of the algorithms.

Table 4.7 also shows that HERR actually performs better than one of Qiskits competitive algorithms, stochastic SWAP. For every benchmark besides QFT4 HERR out performed it. This is primarily due to HERR much shorter routing time overcoming its lack of accuracy compared to stochastic SWAP. Even more interesting that that is that BasicSwap out performed HERR for all QFT benchmarks. The reason for this is the same as the reason HERR performed better than stochastic: the moderate accuracy gains did not outweigh the increased time. That said, whether or not the trade off of increase time efficiency of BasicSwap or HERR is debatable. There were

Table 4.7: Ratio of Accuracy to normalized routing time

Bench mark	HERR	Basic	Sabre	Lookahead	Stochastic
QFT4	0.0772	0.1054	**0.4517**	0.0041	0.0804
QFT7	0.0075	0.0089	**0.0331**	0.0002	0.0057
QFT8	0.0021	0.0039	**0.0076**	0.0001	0.0015
BV4	**2.0392**	0.6502	1.1912	0.1845	0.6102
BV8	0.4508	0.3026	**1.8997**	0.0197	0.3213
Toffoli	0.7926	0.6096	**0.9147**	0.0303	0.5863

not large differences in the routing time for all, so most likely the trade off is not worth it.

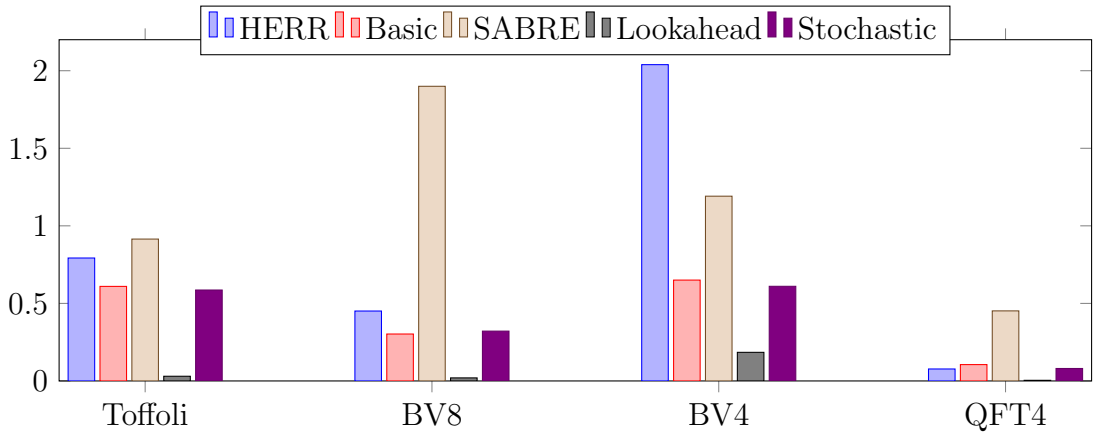


Figure 4.10: Ratio of accuracy to normalized routing time

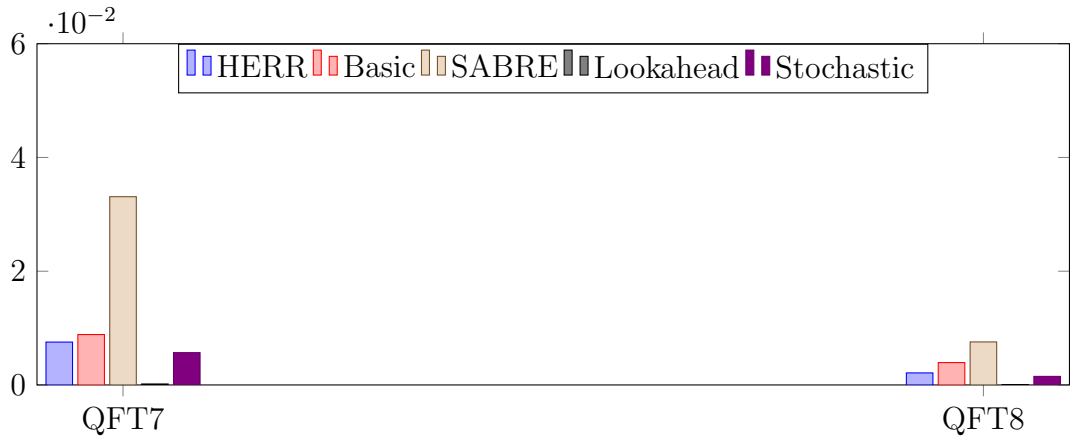


Figure 4.11: Ratio of accuracy to normalized routing time for QFT7 and 8

Chapter 5

Conclusion

5.1 Future Improvements

As section 4 has shown, HERR shows promise based on its gains over Qiskit's Basic-Swap algorithm, but still does not match the accuracy of Qiskit's other algorithms. HERR performs better on some benchmarks, such as the Bernstein-Vazirani algorithm, but performs worse on other such as the QFT. The most likely reason for this is that HERR only focuses on noise adaptive routing by adding extra SWAP gates, and does not attempt to efficiently route the circuit in all other cases. Take for instance SABRE swap. The goal of Sabre SWAP is to reduce the amount of added SWAP gates [6]. If HERR could implement a similar scheme to both avoid noise poor links while also reducing the amount of SWAP gates it uses for normal routing it is possible for large performance gains. So additional work is needed on better characterizing HERRs strengths and weaknesses, as well as developing a hybrid algorithm that can account for those weaknesses.

Another are of improvement is the initial mapping. Tannu and Qureshi show that choosing an initial mapping that avoid poor links can lead to large performance gains in noisy systems [2]. The SABRE swap algorithm also shows that initial mapping has a large role in increasing accuracy, as it can lead to less SWAP gates being needed [6]. Currently, HERR only uses basic initial mapping that maps each logical qubit to its

corresponding physical qubit. If HERR could somehow integrate the initial mapping schemes from [2] and [6] large performance gains could be seen.

5.2 Conclusion

This paper explores noise adaptive routing and proposes a novel solution to the routing problem: HERR (High Error Rate Routing). This implementation differs from previous routing algorithms, as it avoids noisy links to reduce circuit error. This work compares the new algorithm against others used in Qiskit, providing some insight on these previously existing routing algorithms' performance in addition to the new one. Using basic test benchmarks —Quantum Fourier Transform (QFT) [3], the Bernstein-Vazirani (BV) algorithm [9], and a Toffoli gate [3]—, it is observed that HERR is most efficient on high error situations, as expected. While HERR does not outperform Qiskit's existing routing algorithms in all instances, it performed better in smaller and more shallow circuit, namely the four qubit Bernstein-Vazirani algorithm and the Toffoli gate.

Looking at a diverse set of metrics besides accuracy, some additional conclusions can be extracted. Lookahead showed to have extremely high computational cost, — the best performing algorithm in several cases— of 77x Basic as opposed to the modest extra cost of HERR, 1.39x compared to the same Basic case. It was also observed that merely using the number of SWAP gates as a proxy for accuracy may not be the best approach. Some routing approaches targeting to insert a minimal number SWAP gates did not perform as well as HERR, which attempts to improve accuracy through damaged links avoidance. Lastly, the reduced set of test that have been performed already demonstrate that the best routing algorithm is highly dependent on the application circuit that is being targeted.

The first future goal to expand the set of benchmarks to test the routing algorithms. In addition, it is interesting to try to understand what features of a circuit

design make one routing algorithm perform better in each case, given the high variance of results shown in this paper. HERR could potentially also be combined with other routing algorithms besides Basic. Future work could develop guidelines to identify these best suited routing approaches and develop a hybrid algorithm that dynamically select the best approach. Lastly, a comparison against other noise aware existing algorithms will also part of this work's extensions.

Bibliography

[1] S. S. Tannu and M. K. Qureshi, "Not all qubits are created equal: A case for variability-aware policies for nisq-era quantum computers," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 987–999. [Online]. Available: https://doi.org/10.1145/3297858.3304007

[2] P. Murali, J. Baker, A. Javadi-Abhari, F. Chong, and M. Martonosi, "Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers," 04 2019, pp. 1015–1029.

[3] P. Kaye, R. Laflamme, and M. Mosca, *An introduction to quantum computing*. Oxford University Press, 2010.

[4] T. Itoko and T. Imamichi, "Scheduling of operations in quantum compiler," in *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*, 2020, pp. 337–344.

[5] C.-T. Hwang, J.-H. Lee, and Y.-C. Hsu, "A formal approach to the scheduling problem in high level synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, no. 4, pp. 464–475, 1991.

[6] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for nisq-era quantum devices," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1001–1014. [Online]. Available: https://doi.org/10.1145/3297858.3304023

[7] IBM, "Qiskit terra," 2022. [Online]. Available: https://github.com/Qiskit/qiskit-terra

[8] S. Jandura. (2018) Improving a quantum compiler. [Online]. Available: https://medium.com/qiskit/improving-a-quantum-compiler-48410d7a7084

[9] E. Bernstein and U. Vazirani, "Quantum complexity theory," in *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, ser. STOC '93. New York, NY, USA: Association for Computing Machinery, 1993, p. 11–20. [Online]. Available: https://doi.org/10.1145/167088.167097

[10] ——, "Quantum complexity theory," *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1411–1473, 1997. [Online]. Available: https://doi.org/10.1137/S0097539796300921