4-2023

A Survey of Qubit Routing Algorithms

Harrison Barnes
hjb2677@rit.edu

A Survey of Qubit Routing Algorithms

Harrison Barnes

A Survey of Qubit Routing Algorithms

Harrison Barnes

April 2023

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

A Survey of Qubit Routing Algorithms

Harrison Barnes

Committee Approval:

Acknowledgments

I'd like to thank my Advisor, Dr. Sonia Lopez Alarcon, for her assistance in the research, development, and writing of this thesis. I'd also like to thank my family and my fiancée for their support throughout my college.

This work is dedicated to my fiancée Anna

Abstract

Before applications actually run on quantum hardware, the design of the quantum application and its pre-processing take place in the classical environment. Transforming a quantum circuit into a form that is able to run on quantum hardware, known as transpilation, is a crucial process in performing quantum experiments. Routing is a particularly important and computationally expensive part of the transpilation process. The routing process can alter a circuit's accuracy, execution time, quantum gate count, and depth. Several routing algorithms exist that aim to optimize some portion of the routing process. There are many variables to consider when choosing an optimal routing algorithm, such as routing time, circuit accuracy, gate count, circuit execution time, and more. Identifying what routing algorithm is best for a given quantum experiment based on various properties can allow researchers to make informed routing decisions, as well as optimize target aspects of their work. This thesis draws connections between the properties of circuits, hardware, and routing algorithms to identify when a specific routing algorithm will outperform the others.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Quantum computing utilizes properties of quantum mechanics, such as superposition and entanglement, to perform calculations. Quantum computers can potentially perform certain algorithms and operations faster than classical computers. Many scientists, researchers, and companies are working in the field of quantum computing to make advances in various fields. However, quantum computing is inhibited by errors and small computer sizes. This is known as the Noisy Intermediate-Scale Quantum (NISQ) era. Efficient mapping, routing, and scheduling of quantum algorithms onto the NISQ hardware are essential to increasing the efficiency and accuracy of current devices, as well as planning for future hardware implementations. Knowing which routing algorithms are best for a particular problem is crucial to both fully utilizing NISQ devices and preparing for future quantum hardware environments.

1.2 Quantum Routing

Quantum computing is in its early stages, but holds the potential of performing certain algorithms and computations faster or better than classical computers. Much research is being done in the field to make advances in various fronts. However, quantum computing is inhibited by errors and small computer sizes. Efficient transpilation steps, including mapping, routing, and scheduling, of quantum algorithms onto the NISQ hardware are essential to increasing the efficiency and accuracy of current devices, as well as planning for future hardware implementations.

Routing plays a key role in the efficient implementation of quantum circuits on specific quantum architectures and layouts. Quantum circuits represent quantum operations as one-qubit and two-qubit gates. For superconducting qubits like IBM's quantum computers, these qubits need to be physically adjacent to be part of the same two-qubit gate. Therefore, logical qubits used to define the quantum circuit need to be mapped to a set of physical qubits, and then be routed to adjacent physical qubits when they are the target of a two-qubit gate. This problem is challenging and critical for several reasons. First, routing algorithms will add SWAP gates to the overall circuit, introducing additional error and depth. Optimized routing will aim to minimize these parameters. When routing is noise-aware, it can help with noise and error mitigation by routing to less noisy links. Lastly, finding the optimal routing is an NP-hard problem. Thus, various routing algorithms exist that balance the efficiency of the routing with the execution time of the routing algorithm, as finding the most optimal solution is computationally expensive.

Quantum routing algorithms must make compromises by targeting specific metrics and neglecting others. Several metrics could be the target, including routing time, the performance of the routed circuit, the amount of qubits used, the amount of parallelism in the routed circuit, and the error (or accuracy) of the resulting output. On the other hand, multiple variables play a role in this routing problem, such as the size and structure of the circuit, the topology of the quantum computer, the number of qubits used in the circuit, and the noise levels and link quality of the hardware onto which the circuit is implemented.

Several routing algorithms have been proposed in the past. These algorithms target SWAP count, circuit depth, noise, etc. However, there is no survey of all of these algorithms, furthermore, no systematic review of the variables and metrics above has been published. Current experiments target small size circuits due to the reduced hardware resources and the computational challenges of simulation. Understanding how these algorithms will scale in future quantum environments is also a desired insight of this problem. Often times, authors will compare their routing algorithm to one other benchmark algorithm, rather than a holistic review against the currently well utilized routing algorithms.

Knowing which routing algorithms are best for a particular problem, or problem characteristics, is crucial to both fully utilizing NISQ devices and preparing for future quantum hardware environments with more qubits and less noise. This thesis aims at providing these insights through the comparison of five routing algorithms, and systematic variation of problem features and metrics.

1.3 Objective

The objective of this thesis is to determine which routing algorithms are superior when routing different circuits, hardware configurations, and metrics.

Chapter 2

Background

2.1 Quantum Computing

Classical computing utilizes binary bits to perform calculations, where bits are either ones or zeros. *Quantum computing* utilizes qubits to perform calculations, which can be one, zero, or a superposition of both states. A quantum computer is a quantum system and is a controlled quantum-mechanical evolution from an initial state to a final state, which is the solution to a computational problem. When in superposition, the state of the quantum system is unknown. A quantum state can be represented as a state vector, shown in Equation 2.1.

$$|\Psi\rangle = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_m \end{bmatrix} \tag{2.1}$$

The state vector $|\Psi\rangle$ in (2.1) consists of m entries where $m = 2^n$. Note that n is the number of qubits. Thus, as the number of qubits increases, the state vector grows exponentially. $|\Psi\rangle$ can also be represented as a linear expression, as seen in Equation 2.2.

$$|\Psi\rangle = \alpha_1 |\Psi_1\rangle + \alpha_2 |\Psi_2\rangle + \ldots + \alpha_m |\Psi_m\rangle \tag{2.2}$$

$|\Psi\rangle$ in (2.2) is a linear relationship with the basis states, $|\Psi_x\rangle$. Upon measurement of the quantum state, this will collapse to one of the basis states, $|\Psi_x\rangle$, with probability $P(\Psi_x) = |\alpha_x^2|$. Therefore, the act of measuring a qubit collapses the state into sets of zeros and ones in the computational basis. Information about the superposition state can be extracted through multiple measurements in the computational basis (or other bases as needed for quantum tomography). Repeating the quantum circuit and measuring each run's final state allows for the construction of a probability distribution of the final states. In an ideal scenario with enough runs, also referred to as *shots*, and no noise, the probability of finding $|\Psi\rangle$ in state Ψ_x will be $|\alpha_x^2|$.

Quantum algorithms are implemented using *quantum gates* to manipulate one or more qubits. Gates perform computations by altering the superposition state of qubits. Many complex gates exist, but quantum computers utilize a set of *basis gates* that compose all higher-order quantum gates. Basis gates, also called native gates, are set by the quantum hardware, not the programmer. To switch states between two qubits, routing algorithms utilize *SWAP gates* which are implemented with three control-not (CNOT) gates, also known as control-x (CX) gates. Figure 2.1 shows the makeup of a SWAP gate, consisting of three alternating CNOT gates.

CNOT gates invert the state of the target qubit, denoted by the $\oplus$ symbol, when the control bit, denoted by the black dot, is in a *one* state. The SWAP in Figure 2.1 will switch the states of the two qubits x and y. SWAP gates have many applications, such as checking for equality, swapping states for routing, and are used in various algorithms.

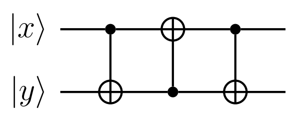
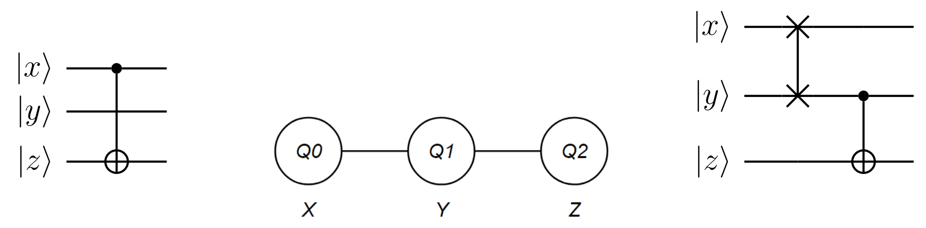


Figure 2.1: Decomposition of a SWAP Gate into CNOT Gates

2.2 Quantum Compilation

With some parallelism with classical hardware description languages, quantum circuits must be compiled from code into a hardware implementation. Quantum compilers take the code, which represents a circuit for generic quantum hardware, and compile it into a circuit that will run on the target quantum hardware. This process requires the transformation of the circuit to meet the real hardware constraints present in the target device. For example, not all basis gates are the same for every quantum device. The compiler decomposes the circuit into the basis gates by breaking high-level gates into their native components. Furthermore, logical qubits, which exist solely within the code, must be mapped to physical qubits on hardware. This has two issues: First, if a circuit cannot fit on a device, changes must be made to the circuit so that it will fit on the device or a larger device must be selected if available. This limits the size of problems that can be solved on a device, as the number of qubits is a major limiting factor. Second, qubits can only interact with physically adjacent qubits. Since almost all quantum algorithms require some form of a two-qubit gate, routing must occur. *Routing* places SWAPs between qubits with the goal of moving qubit states around for interaction. Routing is required when a gate is applied on non-adjacent qubits, as shown in Figure 2.2.

The quantum circuit in Figure 2.2a shows a CNOT gate acting on qubits x and z. If the quantum compiler maps the logical qubits to the physical qubits in the 3-qubit processor shown in Figure 2.2b. When mapped to the processor, the CNOT gate working on x and z cannot be performed since the corresponding physical qubits *Q0* and *Q2* are not adjacent. Routing rectifies this issue by adding a SWAP gate between either x and y, as shown in Figure 2.2c. This allows the CNOT to be performed on the adjacent qubit pair: *Q1* and *Q2*. Another option is to SWAP y and z and perform the CNOT on x and y. As circuit complexity increases, the possible routing decisions also increase, adding to the complexity of routing the circuit for the target hardware. Efficient routing algorithms are time efficient and or accurate. Since routing is an NP-complete problem, finding the optimal routing solution is time intensive. Thus, algorithms utilize different heuristics that balance routing time and accuracy.



(a) Unrouted CNOT Circuit **(b)** 3-Qubit Processor **(c)** Routed CNOT Circuit

Figure 2.2: Routing a simple circuit on a theoretical 3-Qubit quantum processor to satisfy two-qubit gate adjacency

2.3 Error and Noise

As the name suggests, devices in the NISQ era are noisy. Noise in a quantum system degrades the accuracy of the operations and results. Essentially, noise creates errors as the circuit progresses. If enough error is accumulated, the correct results will not be extractable from the probability distribution. Quantum noise puts a limit on how deep circuits can go and how many operations can be performed. *Circuit depth* refers to the number of layers of gates. Depth is important as physical qubits will begin to suffer state degradation after a period of time. Gate errors are another source of error. The application of a gate onto a qubit has a specific error associated with it. Due to this, the more gates applied, the more error introduced into the system. Coherence errors and gate errors limit the realistic depth of the circuit.

Two-qubit gate errors are dependent on the topology and noise levels of the target hardware. Links between qubits have noise associated with them. Applying a gate on a link is subject to the noise present on that link. Furthermore, not all links have the same noise rate.

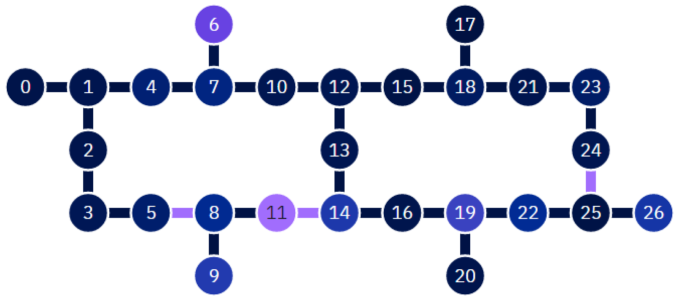
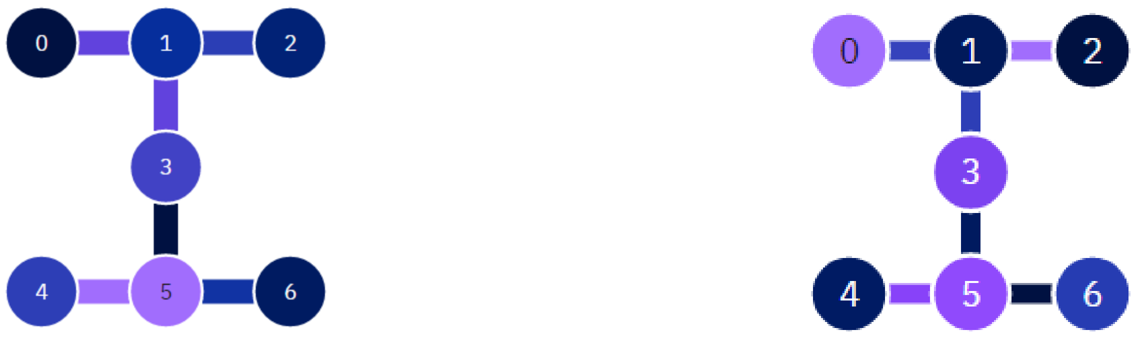


Figure 2.3: Link Errors for IBM Kolkata's 27-Qubit Falcon r5.11 Quantum Processor. Link colors correspond to CNOT link error, where lighter links have a higher error rate than darker links.

Figure 2.3 shows qubit operating frequencies and link errors between qubits for IBM Kolkata's 27-Qubit Falcon r5.11 quantum processor [1]. The qubits, represented as colored circles with qubit numbers in them, are colored based on their operating frequency. Darker blues represent lower operating frequencies, while lighter colors, such as purple, represent higher frequencies. However, the more important information is the CNOT connection error. IBM defines their link error as the expected error induced by performing a CNOT operation across a link. In this instance, a darker color represents a lower error, while a lighter color indicates a higher error. Thus, an efficient routing algorithm may aim to utilize low-error links as much as possible. This type of algorithm is known as *noise-aware routing*. A noise-aware routing algorithm must be flexible, as not all topologies have the same error rate, as shown in Figure 2.4



(a) IBM Oslo **(b)** IBM Jakarta

Figure 2.4: Different errors for different 7-Qubit IBM Falcon r5.11H Quantum Processors. Link colors correspond to CNOT link error, where lighter links have a higher error rate than darker links.

Figure 2.4 shows the CNOT connector errors for the IBM Oslo and Jakarta 7-qubit Falcon r5.11H quantum processors, as reported by IBMQ API [1]. The difference between the devices is noticeable, as no two corresponding links are equivalent across the devices. For example, the link from Q1 to Q2 has a low CNOT error on IBM Oslo yet has a high error on IBM Jakarta. A flexible noise-aware routing algorithm must take into account the error map of the target device. Furthermore, the difference in topologies between Figure 2.3 and Figure 2.4 requires a routing algorithm to have flexibility in its decision-making process. Finally, the noise between qubits varies day-to-day, requiring noise-aware routing algorithms to adapt as noise changes in a quantum processor.

2.4 Existing Routing Algorithms

IBM Qiskit is an open-source quantum development kit that allows users to develop quantum circuits, perform quantum compilation, run simulations, and run circuits on IBM's quantum hardware [2]. Qiskit's transpiler translates the quantum circuit code into a hardware-executable circuit for the target device. The transpiler routes the circuit for the target hardware, selecting an algorithm based on user specification.

Qiskit comes with five routing algorithms: Basic Swap, Lookahead Swap, Stochastic Swap, BIP Mapping, and SABRE. An alternative noise-aware routing algorithm will also be explored.

2.4.1 Basic Swap

Basic Swap is a simple algorithm that trades time efficiency and simplicity for optimization. If a gate is applied to non-adjacent qubits, SWAP gates are added along the shortest path between the two qubits [3]. This algorithm is computationally cheap, easy to implement, and works well as a fallback routing method for other complex methods. However, Basic Swap does not take link noise, SWAP parallelism, and future SWAPs into account. Thus, the final routing may not be optimal.

2.4.2 Lookahead Swap

Lookahead Swap generates a directed acyclic graph (DAG) for the quantum circuit, which is then used to create an initial mapping that allows the most reuse of qubits without SWAPs [4]. Then, Lookahead Swap will select the top N possible SWAPs for a gate and repeat the analysis M times for each of the N possible SWAPs, where N and M are search parameters set by the user. Lookahead then selects the best of the possible SWAPs and moves on. This algorithm attempts to minimize the number of SWAP gates, which can increase accuracy and lower depth. However, this algorithm generates N^M layouts when analyzing a gate, which is computationally intensive and may result in high routing times.

2.4.3 Stochastic Swap

Stochastic Swap which generates a DAG that is broken into layers [5]. Random permutations are used to generate SWAP configurations, which are compared to find the best SWAP configuration for a particular layer. Optimally solving each DAG layer increases the accuracy of stochastic at the potential cost of high run times. Furthermore, restricting the search to a single layer of gates may ignore more optimal solutions that span multiple layers, such as those found in Lookahead Swap. However, this allows stochastic the potential to decrease the total depth of the circuit.

2.4.4 BIP Mapping

BIP Mapping converts the circuit to a binary integer programming (BIP) problem [6]. Potential SWAPs are given a weight based on the user-specified algorithm priority. Different priorities will perform different mathematical operations on the weights in the BIP problem using Python math libraries. BIP Mapping offers some versatility through the implementation of algorithm priorities. BIP Mapping defaults to a balanced priority but can either prioritize the depth or the gate error. Prioritizing gate error turns BIP Mapping into a noise-aware routing algorithm, as it will utilize the IBM backend link errors to inform routing decisions. BIP Mapping has some drawbacks; the number of virtual qubits and physical qubits must match, and the main libraries struggle with double-digit qubit counts. Furthermore, the complexity of the algorithm results in high execution times, along with dense, abstract source code with most of the work hidden in non-Qiskit libraries.

Due to the extremely limiting qubit maximum of BIP Mapping, the algorithm is not explored further in this work.

2.4.5 SABRE

SABRE generates a DAG for the circuit and breaks it into layers [7]. If the front layer of gates cannot be applied directly, SABRE Swap will search and add SWAP gates. A heuristic cost function is used to restrict the pool of potential SWAPs to locations near the current layer, adding the best one of the generated SWAPs. The front layer is removed from the DAG when all gates can be applied, and the next layer becomes the front layer. Qiskit alters the algorithm by doing multiple passes with different seeds and selecting the best [8]. SABRE Swap does not aim to find the optimal solution, which can save execution time at the cost of CNOT count and accuracy. However, by restricting the search to layers near the front, some parallelism is built into the algorithm, along with the ability to tune the cost function if needed.

2.4.6 High Error Rate Routing

High Error Rate Routing (HERR) is an additional algorithm, not included in the Qiskit tools, but that will be referred to in this work [9]. HERR is a noise-aware algorithm that routes around noisy links if the extra SWAPs will yield an overall lower error [9]. If no such path exists, HERR defaults to the Basic Swap algorithm by following the shortest path. HERR has the potential to produce more accurate results at cheaper run times, due to it being a slightly more complex Basic Swap. However, some of Basic Swaps' drawbacks carry over to HERR due to its dependency as a fallback. Furthermore, reliance on low-error links may create bottlenecks in certain noisy devices. HERR handles layout edits differently than basic does, which may result in lower accuracy due to the variance in link errors. Finally, the increased SWAP count may increase the depth, which can harm the efficiency of already deep circuits.

2.5 Existing Research

HERR was bench marked against four other routing algorithms: Basic Swap, Stochastic Swap, Lookahead Swap, and SABRE [9]. Three benchmark circuits were chosen: a quantum Fourier transforms (QFT), the Bernstein-Vazirani algorithm (BV), and the Toffoli gate. All of these benchmarks were routed and performed on one of IBM's Falcon r5.11H quantum processors, as well as two theoretical grid maps with four and eight qubits. HERR's benchmarking process utilized two random noise models, 1-10% noise for a typical system, and 1-20% noise for a substantially noisy system.

HERR was evaluated on three metrics: accuracy, CNOT count, and routing execution time. Accuracy measures the percentage of time that the routed circuit produces the correct result. CNOT count is the number of CNOT gates present in the circuit. HERR claims that the CNOT count is not as vital as accuracy but includes it as a metric anyways, as many algorithms use CNOT count as their main evaluation metric. Finally, the execution time measures the speed of the routing algorithm. These metrics can be used to determine the scalability and feasibility of a routing algorithm and are used to measure up HERR against other algorithms.

SABRE tests itself against one other algorithm through numerous benchmark circuits. In terms of metrics, SABRE strictly focuses on CNOT count and execution time. The authors of SABRE correlate an increased CNOT count with increased error [7]. Both SABRE and HERR aim for similar goals, but the nature of their algorithms results in the authors emphasizing different metrics. HERR puts the most emphasis on increasing accuracy rather than CNOT count, as HERR intentionally adds CNOTs to decrease noise. Alternatively, SABRE focuses on the CNOT count as the sole accuracy metric, due to the author's view that increasing the CNOT count decreases accuracy. Furthermore, SABRE only compares itself to one other algorithm [10], and does not mention the noise model used.

Chapter 3

Methodology

3.1 Benchmarks and Metrics

A series of experiments were developed to determine the efficiency and usefulness of routing algorithms under various conditions. The selected routing algorithms for testing are basic swap (BSC), lookahead swap (LKHD), stochastic swap (STCH), SABRE, and HERR. The test conditions included evaluation metrics, hardware characteristics, and circuit selection. The selected metrics aim to quantify the accuracy, parallelism, and speed of the routing process for each routing algorithm. The data from the metrics was then evaluated for each test scenario corresponding to a specific set of hardware characteristics and circuits. To reach meaningful conclusions based on experimental results, a range of metrics, characteristics, and circuits were selected.

3.1.1 Accuracy, Depth, & Timing

The routing algorithms under review are evaluated according to three metrics: accuracy of results, depth of the circuit, and the time it takes to transpile the circuit.

Accuracy is not a straightforward metric as there are multiple ways to determine how accurate a circuit is. In this thesis, two accuracy metrics are utilized. First, the number of CNOT gates added by the routing algorithm is determined after routing. This is referred as the *delta CNOT*, or the change in CNOT count over the original, pre-transpilation circuit. Each CNOT gate introduces additional error into the states of the relevant qubits. A circuit with more CNOTs will generally have worse accuracy than a circuit without CNOTs.

Noise-aware routing algorithms have proven that adding additional CNOT gates may be beneficial, reducing the overall error by avoiding links with large errors [9]. Thus, CNOT delta is not always a definitive measure when comparing the accuracy of two routing algorithms. Thus a second accuracy metric was implemented. Each routed circuit would be simulated to obtain a probability distribution. For basis encoded circuits, the accuracy is defined as the percentage of shots that produced a correct result. Since routing directly affects the gate count, the same circuit will have different accuracies after transpilation. For amplitude encoded circuits, calculating the simulated statevector and comparing it to the expected statevector would provide usable accuracy metric. The expected statevector is the statevector of the circuit without any noise. Comparing the simulated and expected statevectors would utilize Manhattan distance due to its viability with high dimension spaces [11]. However, due to the high run time and high number of required shots for large QFT circuits, this metric will not be used in this work. Only basis encoded algorithms will be examined using % shot accuracy

IBM quantum computers perform quantum operations layers at a time. Thus, each qubit's gates act in parallel with other gates on that layer. The number of layers in a quantum circuit is known as the depth of the circuit. Reducing circuit depth reduces the number of layers needed to evolve the quantum state from the initial state to the final state. Furthermore, for circuit simulation, reducing the quantum depth reduces its memory requirements and run time. Thus, *depth delta* is measured for each circuit by calculating the additional number of layers in the circuit's DAG compared to the unrouted circuit. This allows for routing algorithms to be compared in terms of the parallelism of their routed implementation.

Performing routing algorithms on large quantum circuits devices takes a significant amount of time. For each algorithm, the *transpile times* are measured and recorded. This metric shows which algorithms perform routing more efficiently on a given hardware with a given circuit. Transpilation time is measured instead of just routing time since the routing process affects the transpilation process. Measuring only the routing time does not reflect the routing process' impact on the decomposition, optimization, etc. steps of the transpilation process. Measuring the transpilation times allows a developer to trade off increased accuracy for increased routing times.

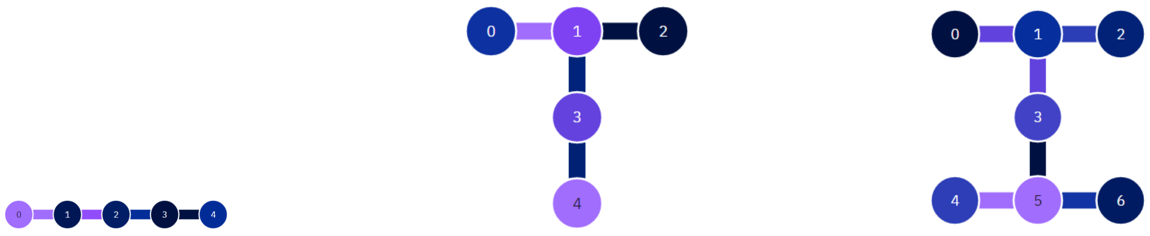
3.1.2 Hardware characteristics

IBM offers a wide range of quantum computing hardware architectures. Each device varies in topology and qubit count. Table 3.1 shows the devices used in this work and the respective qubit counts and topology.

Table 3.1: IBM quantum computer characteristics

Device	# Qubits	Topology
Manila	5	5-
Quito	5	5T
Oslo	7	7H
Guadalupe	16	16X
Toronto	27	27X
Washington	127	127X

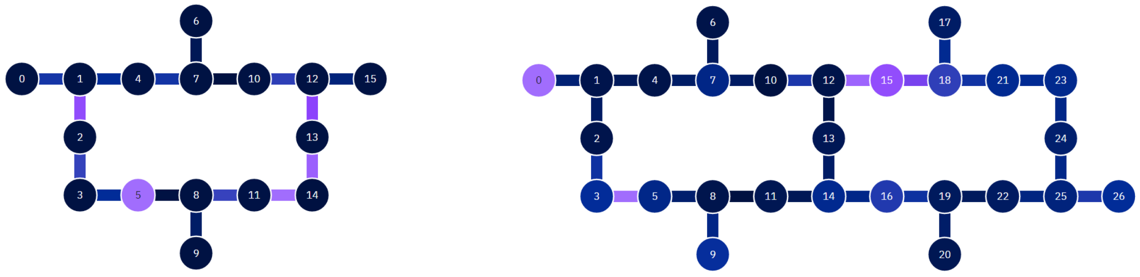
The devices listed in Table 3.1 are the IBM Quantum Computers utilized in this thesis. The devices are named after specific cities in the world, making them easily identifiable. The topology of the five and seven qubit machines differs from that of higher order computers. Figure 3.1 shows the topology of the Manila, Quito, and Oslo IBM quantum computers.



(a) IBM Manila (b) IBM Quito (c) IBM Oslo

Figure 3.1: Coupling maps for three IBM quantum computers: Manila, Quito, and Oslo

The coupling maps in Figure 3.1 are three IBM quantum computers used during testing. These are simple coupling maps for simple, low qubit quantum computers that IBM offers for free to all IBM Quantum users [1]. Figure 3.2 shows the coupling map for the 16 and 27 qubit devices.



(a) IBM Guadalupe (b) IBM Toronto

Figure 3.2: Coupling maps for two IBM quantum computers with hexagon topology

The quantum systems in Figure 3.2 are part of IBM's paid program as they are able to handle larger qubit circuits. Guadalupe and Toronto begin the trend of IBM's topology for all computers larger than seven qubits. Guadalupe serves as a building block for IBM's hexagon topology, where all subsequent systems use the hexagon as the basic qubit building block configuration. Toronto, the next largest quantum computer, utilizes two hexagons. For example, IBM Washington is a 127-qubit quantum computer consisting of 18 hexagons. IBM Washington will be used for detailed analysis of trends involving routing's impact on depth and CNOT count.

3.1.3 Circuit Selection

There are two factors that play into circuit selection. Primarily, complex circuits with more gates and higher depth require more intensive routing and transpilation while also greatly reducing the accuracy of the evolved statevector. The *gate complexity* of the circuit is the big-O complexity of the relationship between the number of two-qubit gates in the circuit and the number of logical qubits n. Larger gate complexities are more expensive for routing algorithms. Another major factor is the *encoding* of the output state. Two types of encoding exist: *basis encoded* and *amplitude encoded.* Basis encoding algorithms encode their solutions in the most probable basis state, such as Grover's algorithm [12]. Amplitude encoding algorithms encode their solutions in the amplitude of each basis state, such as HLL [13]. Table 3.2 shows the selected quantum circuits with their gate complexity and state encoding.

Table 3.2: Quantum algorithms selected for testing

Quantum Algorithm	# Gate Complexity	State Encoding
QFT	$O(2^n)$	Amplitude
Deutsch–Jozsa	O(n)	Basis
W-State	O(n)	Basis
GHZ State	O(n)	Basis

The quantum circuits listed in Table 3.2 are used with various qubit counts across different hardware tests. The circuit test set includes quantum fourier transforms (QFT), Deutsch-Josza (DJ), W-State (WS) and Greenberger–Horne–Zeilinger State (GHZ). All quantum circuits are obtained from the Munich Quantum Toolkit Benchmark Library [14]. These circuits can show how a specific routing algorithm handles gate complexity and state encoding schemes. QFT covers amplitude encoded algorithms, as well as complex circuits. DJ, WS, and GHZ are all basis encoded with varying level of complexity. These three were selected to examine how routing algorithms handle various complexity of basis encoded circuits. This knowledge can inform developers which routing algorithm they should use based on their target quantum algorithm. This circuit set is intentionally small, they are easy to analyze and form trends, especially for accuracy, a metric that is limited to smaller, simpler circuits.

3.2 Experiment Selection

Using the selected metrics, hardware, and quantum circuits, a set of quantum experiments can be created to accurately map the efficiency of routing algorithms under various conditions. Three sets of experiments were defined: topology-based, occupancy-based, and noise-based. Topology-based experiments will focus on how routing algorithms perform on various IBM quantum architectures. Occupancy-based experiments will examine how routing algorithms perform at various occupancy levels on the same topology. A quantum computer's *occupancy* is defined as the number of logical qubits in the quantum circuit divided by the number of physical qubits on the device. Finally, noise-based experiments will alter the noise map of the quantum hardware to examine how noise affects the routing algorithms' performance.

3.2.1 Topology

The various circuit topologies listed in 3.1 were tested with the full suite of circuits listed in 3.2, and are evaluated using all metrics. All circuits are performed at max occupancy for the hardware. Thus, only differences in topology should affect the performance of the routing algorithm. This test is meant to establish a baseline understanding of the effectiveness of the routing algorithms, as this test encompasses the most circuits and hardware.

3.2.2 Occupancy

Not all circuits will utilize the entire coupling map of a quantum computer. A circuit with more logical qubits requires more physical qubits to execute. Consequently, higher occupancy requires more routing while having fewer ancilla qubits for routing purposes. Occupancy is an important metric as certain routing algorithms may perform well at certain occupancies but not at others. Thus, occupancy is varied on larger qubit circuits to determine how routing algorithms deal with ancilla qubit constraints when performing routing on a quantum circuit. The hexagon topologies will be used, along with one circuit for each encoding type: QFT and DJ. These circuits were selected to cover both encoding types, as well as provide a circuit complex enough such that routing will impact the accuracy of the evolved statevector.

The occupancy test also includes CNOT and depth analysis for routing QFTs onto IBM Washington. The top algorithms will be selected and compared to determine how they could perform on future hardware with larger architectures.

3.2.3 Quantum Noise

Quantum computers possess various physical qualities regarding their qubits and qubit connections. Noise affects the accuracy of circuits by affecting the application of quantum gates. Routing algorithms are most affected by the CNOT link error, as SWAP gates are required to perform routing, resulting in a decreased accuracy (generally). Currently, quantum computers are noisy, but this will improve with time. Thus, testing how accurate routing algorithms are on a noise-reduced quantum computer is important. Furthermore, the quality of a link can degrade, resulting in some links being far less accurate than other links. Testing how a routing algorithm will perform when a small percentage of links have much higher error than other links can aid in routing algorithm selection for quantum hardware with bad links in the system.

Quantum noise is tested by using *adjusted overall noise* and the number *bad links*. Adjusted overall noise is a quantum test where the entire noise map is scaled by a scalar value. For example, testing with an adjusted overall noise of x0.1 would multiply every link's CNOT error by 0.1, resulting in lower error. This test will show how algorithms perform on current noise levels and future reduced noise levels. For this test, DJ circuits of varying size were routed and simulated on IBM Guadalupe and Toronto. A bad link quantum experiment is one where a set number of qubit links are altered to be bad links. A bad link is defined as a link with a CNOT error $> 1.0e-1$. The majority of qubit links on IBM's newer processors have link errors ranging from $1.0e-3 \longrightarrow 9.0e-2$ [1]. For simplicity, all bad links will be set to $2.5e-1$ CNOT error, significantly higher than that of IBM's nominal noise range. The bad links experiments were simulated on IBM Toronto using 12-qubit and 16-qubit DJ circuits, with one to eight bad links for each circuit. The purpose of this experiment is to test how a link with poor error can be avoided by noise-aware routing. Future hardware with less noise may have some links that reach current-day noise, thus becoming a bad link.

3.3 Limitations

All experiments are performed on RIT's Research Computing (RC) [15]. Despite the processing power available, quantum simulation is a computationally expensive procedure, especially for larger qubit simulations. Performing enough shots to determine accuracy for a basis encoded circuit with minimal solutions is simple. However, for an amplitude encoded circuit like QFT, the large solution space requires many more shots to build the probability distribution. As such, DJ is used for all accuracy tests, as it is basis encoded with one solution. DJ is also in a sweet spot where noise does affect the output, but not so much so that routing a large DJ circuit cannot improve the accuracy a meaningful amount. All 27-qubit topology tests will not have accuracy metrics performed due to the high computation requirements for even the simplest circuits.

Chapter 4

BAROQUE

BAROQUE (Blueprint for Assembling the Runtime Operations of Quantum Experiments) is a fancy name for a tool that responds to a simple need: scripting quantum experiments. As previously discussed, there are various pieces of quantum experiments that are altered to gather data, namely hardware, circuit, metric, etc. Varying these parameters resulted in hundreds of different experiments, amounting to hundreds of hard-coded lines of code that would require tedious minimal changes from one to the other.

BAROQUE was originally created by the author to script quantum experiments for data collection related to this work, but quickly grew to become a modular blueprint that encapsulates IBM Qiskit functions. This encapsulation solves two issues. First, some information from IBM is complex to parse into a usable form for Qiskit. Second, Qiskit's documentation is lacking in many areas that are critical for researchers running complex tests beyond the "Qiskit Textbook".

4.1 BAROQUE Background

BAROQUE is built on, and adds upon Qiskit's existing functions. Qiskit is an open-source software development kit (SDK) created and maintained by IBM that provides tools to work with quantum computers. Developers and researchers can use Qiskit for simulation and development of pulses, circuits, and other quantum applications

[2]. Qiskit is widely used in the quantum computing research community due to the wealth of functionality, ease of use for beginners, and open-source nature. IBM's Qiskit allows developers to run programs on both quantum simulators and real quantum hardware. Qiskit also provides users the ability to create quantum circuits, use common quantum algorithms, and explore various analysis methods. However, Qiskit has some shortcomings, and tools could not be found to remedy the specific shortcomings encountered during data collection.

4.1.1 Qiskit Development Environments

IBM provides two browser-based quantum development environments. IBM Quantum Composer [16], provides the user various tools to synthesize a quantum circuit without knowledge of simulation or transpilation. In fact, the ability to drag and drop gates onto qubits removes the need to know quantum assembly language or Qiskit gate operations and functions. Quantum Composer is for entry level developers, those learning the basics of quantum computing, and developers simulating small circuits without wanting to set up a full environment or Jupyter Notebook. Due to its simplicity, Quantum Composer is limited in its functionality and range of experiments it can perform. Quantum Composer is best used for small experiments, or for learning.

IBM's Quantum Lab is the other browser-based quantum development environment [17]. Quantum Lab is a Jupyter Notebook environment with access to Qiskit. A Jupyter Notebook can be scheduled to run on an IBM simulator, IBM quantum system, or simply run in the Jupyter environment. Quantum Lab does not offer any user interface development of quantum circuits. However, Quantum Lab allows the developer to create complex code in an browser-based integrated development environment (IDE) using Jupyter Notebook.

The Qiskit SDK is also available to download and integrate into an IDE such as PyCharm. The SDK gives full functionality to developers within an IDE. However, developers using Qiskit in an IDE must perform extra steps that would normally be handled by Quantum Composer or Quantum Lab, such as logging into IBM Quantum.

4.1.2 Qiskit Documentation and Metrics

IBM provides a Qiskit Textbook containing well-documented examples of how to use Qiskit for various simple tasks, algorithms, and circuits [2]. Gathering niche information on the fly about quantum compute resources for simulation is not well documented, nor are there many examples on third party sites such as Github and Stackoverlow.

In addition, Qiskit relies heavily on string constants that can be hard to track through its documentation. The three most common string constant groups are backends, quantum gates, and routing algorithms. The first group, backends, refers to both IBM quantum systems and IBM quantum simulators. The second group refers to the basis gates IBM systems decompose circuits into. The last group consists of the different routing algorithms Qiskit has available.

Last, Qiskit does not provide some specific metrics that are important for researchers. For example, the output of a simulation can be collected as a probability distribution of measurements in the computational basis, but Qiskit does not provide a simple, readily available metric of the quality of that solution. The researcher will have to then manipulate this probability distribution to come up with the best metric of its quality. Here, it is important to note that the best metric may vary depending on whether the output is basis encoded vs amplitude encoded.

4.2 BAROQUE Design Goals and Targets

BAROQUE was developed with three goals in mind:

- to simplify interfacing with IBM's Quantum API and quantum systems,

- to provide commonly used metrics,
- to create a modular program with user modification in mind, and
- to implement alternative routing algorithms for easy use in Qiskit experiments, particularly HERR

The first three goals required modifying specific parts of the quantum experiment workflow to simplify, atomize and enhance it without compromising the integrity of the targeted process.

4.2.1 Generic Quantum Experiment Design Goals

A *generic quantum experiment (GQE)* is defined as a program that takes a quantum circuit, transpiles it onto the desired backend, and simulates the circuit on that backend.

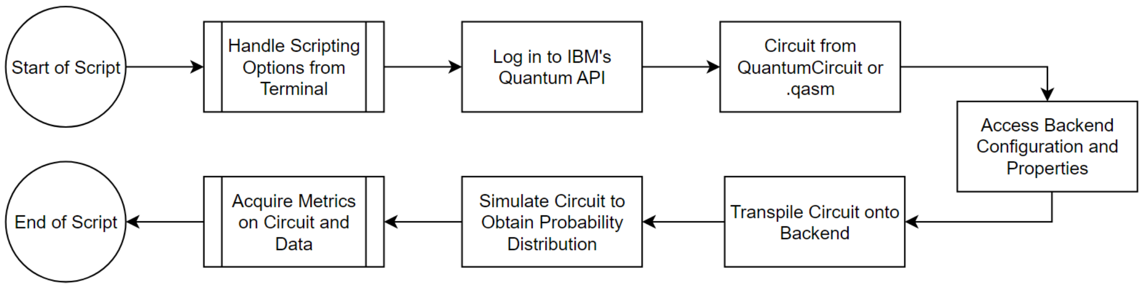

Start of Script
Handle Scripting Options from Terminal
Log in to IBM's Quantum API
Circuit from QuantumCircuit or .qasm
Access Backend Configuration and Properties
Transpile Circuit onto Backend
Simulate Circuit to Obtain Probability Distribution
Acquire Metrics on Circuit and Data
End of Script

Figure 4.1: Generic quantum experiment workflow processes. Double vertical lines indicate optional steps. These processes will allow a script to test a quantum circuit using Qiskit.

Figure 4.1 shows a program flow diagram for a GQE performed in an IDE. Steps such as scripting and acquiring metrics are optional thus they are marked in the flow diagram. This GQE workflow includes processes that almost all experiments will need to go through, regardless of the specific combination of metrics, circuit and backend chosen for the experiment. For GQEs, BAROQUE in particular has four design targets:

- Simplify the log in process into the IBM's Quantum API,
- Facilitate the Backend selection, and access to its qubit map and calibration data,
- Facilitate the simulation of the circuit to obtain final results in the form of a probability distribution,
- Acquire and report a variety of metrics,

A design target is a part of the GQE that BAROQUE aims to encapsulate, simplify, and provide support for. These processes are either not well documented, consist of complex yet reusable code, or a mix of both. *For example:* the configuration and properties of the backends is well documented for some features but not for others. Constructing the CX gate error map is complex, but can be reused without changing the code, for any backend. Thus, BAROQUE targets this step as it removes the need to read scattered pieces of documentation and provides a reusable code-base for any backend.

Acquiring metrics for analyzing the probability distribution is an optional step, yet it is a design goal, and is thus handled by BAROQUE through two means. First, BAROQUE provides a variety of useful metrics for our research, such as accuracy, state vector distance, depth, and CX gate count. Second, BAROQUE implements these main metrics by providing a more generic version of some of them. For example, the CX gate count function uses a generic gate count function. This generic function allows the user to count the standard CX gate, or a different quantum gate, should the need arise. This flexible feature of BAROQUE is depicted in figure 4.2. BARROQUE's metricCountCX is simply modified with the desired gate to count, without having to access cumbersome IBM's Qiskit code. Common gate strings are provided by BAROQUE too, saving the user from locating Qiskit's list of gate string representations.

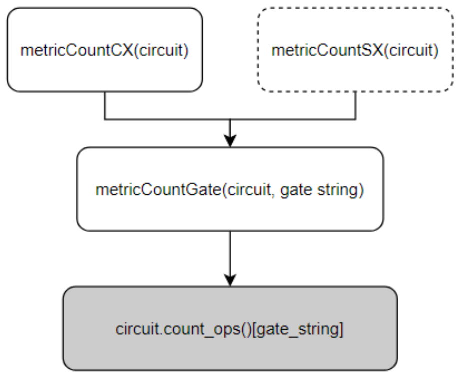

metricCountCX(circuit)
metricCountSX(circuit)
metricCountGate(circuit, gate string)
circuit.count_ops()[gate_string]

Figure 4.2: Dependency diagram showcasing BAROQUE's ability to user's to easily add their own metrics. The original Qiskit code (gray) is hidden from the user, allowing them to make simple additions to their scripts without directly using Qiskit.

The generic CountGate metric shown in Figure 4.2 can be tuned to count the standard CX gate, or any other gate that the researcher requires. The raw Qiskit code [18], shown in gray, is never modified by the user/researcher. All the user needs to do is examine metricCountCX()'s short code and change the gate string to match their desired gate.

BAROQUE does not treat some parts of the GQE workflow as a design target. Qiskit provides a function to convert a quantum assembly file, .qasm, to a Quantum-Circuit object, and the process is well documented. Transpilation is often used in Qiskit's Textbook, and the parameters used are not difficult to decode from documentation. Functions such as these cannot be further encapsulated as Qiskit makes transpilation and circuit implementation coming from an existing benchmark simple for the user.

4.2.2 Alternative Routing Implementation

The fourth and final design goal does not satisfy a specific design target of a GQE workflow. Standard Qiskit's available routing algorithms are: basic swap, SABRE, Lookahead, and Stochastic [3, 19, 20, 5]. HERR works alongside with basic swap. BAROQUE encapsulates HERR and basic swap sections into one routing and transpiling function. This allows users to use HERR as they would use SABRE, Stochastic, etc., expanding their repertoire of routing algorithms. Through this portion of the code, users have the possibility of also adding their own routing algorithms when need be.

4.2.3 Strings and Metrics

Qiskit functions often contain a parameter that requires a specific string chosen from a set of possible strings. For instance, when selecting a backend system to simulate, Qiskit requires a string such as "ibm_oslo" or "ibmq_quito". These fields that can be selected from numerous choices are backends, quantum gates, and routing algorithms. BAROQUE provides a single place that holds commonly used string constants to save developer time in reading Qiskit documentation to locate the exact string used by Qiskit.

BAROQUE simplifies the analysis process by providing and encapsulating the code to common metrics used to analyze a quantum circuit. The more commonly used metrics include circuit depth, gate count, routing time, and transpilation time.

BAROQUE also includes two accuracy metrics: simulation accuracy and statevector norm.

Simulation accuracy requires the user to provide a list of basis states that qualify as an expected correct output, referred to as *answers*. Simulation accuracy determines what percent of shots yielded answers, and returns that percentage. Simulation accuracy is effective for basis encoded experiments. For amplitude encoded experiments, statevector norm is used. First, the developer must provide an expected state vector or generate one. BAROQUE can generate an expected state vector by running the circuit on a noise-less simulation to obtain the expected state vector, or an exact state vector can be provided by the user. Once the expected state vector is obtained, BAROQUE calculates the L1 and L2 norm, returning both in a tuple. This allows the user to select their preferred metric for statevector norm when determining the accuracy of their experiment.

4.3 Example Implementation

To showcase BAROQUE's ability to simplify code segments, an example implementation was created. The original source code is a test program used to benchmark ARA's accuracy with other routing algorithms executing a QFT circuit. BAROQUE was used at applicable spots to reduce the lines of code, simplify functions, and make the program more readable. Lines of code (LOC) will be used as a metric to show the raw reduction in number of lines. LOC does not include white space or comments. It does include imports, function definitions, etc.

Table 4.1: Lines of code per program implementation

Implementation	Lines of Code
Original	102
Baroque w/ ARA	74

Table 4.1 shows the lines of code for the QFT accuracy benchmark based on the specific implementation: original, BAROQUE, and BAROQUE with ARA. The original version consists of 102 LOC, yet BAROQUE simplifies that to 74 LOC. The majority of these reduced lines are in experiment set up.

The advantages of BAROQUE go beyond the LOC metric. The original implementation hard-coded the coupling list, coupling map, error map, etc. as shown in Listing 1 below.

While the original implementation in Listing 1 works without any errors, it is not reusable nor is it necessary. IBM provides these data fields for every quantum system, thus creating a generic container for said information, as seen in Listing 2, increases reusability and readability.

```
-----------------------------------
// Listing 1 - Original set up code


provider = IBMQ.load_account()
backend = provider.get_backend('ibm_oslo')
basis_gates = backend.configuration().basis_gates
osloCouplingList =
    [[0, 1], [1, 0],[1, 2], [2, 1],
    [1, 3], [3, 1], [3, 5], [5, 3],
    [4, 5], [5, 4], [6, 5], [5, 6]]
osloCouplingMap = CouplingMap(osloCouplingList)
-----------------------------------
```

Note that the Oslo coupling list is hardcoded as an array in Listing 1. Pulling the list from IBM would require significantly more LOC, which BAROQUE encapsulates.

```
----------------------------------
// Listing 2 - BAROQUE IBM container code


q_container = IbmInterface.IbmqInterfaceContainer (
   ibmq_api_key,          CommConstants.OSLO_SYS_STR)
---------------------------------
```

The code in Listing 2 is simpler to implement for the user as they do not need to perform the functions of the original code to parse backend data into a usable form. Furthermore, changing the backend system from IBM Oslo to another system would only require the system string to be altered. In the original code listing, the coupling list would need to be redone to match the new system. Another LOC reduction comes from the metric calculation. The original program simulates the circuit, gets the counts, and performs math to determine the accuracy. BAROQUE does all of that with a single function call, reducing the complexity of the main code.

When looking at the main program alone for a small experiment, the LOC reduction is clear. BAROQUE simplifies common processes in a GQE, while promoting reusability by providing functions to replace processes required to set up a GQE. This is a simple implementation example, however the general findings show that any GQE can be simplified due to the common occurrence of the targeted processes. The reduction of LOC will vary from case to case.

4.4 Usage in Data Collection for Thesis

BAROQUE was originally developed to assist in the collection of data for analyzing routing algorithms for this thesis. Scripts were made to automate quantum experiments using BAROQUE to assist in:

- Acquiring backend information on the quantum processor simulated for the experiment
- Simplifying string usage for backends, quantum gates, etc.
- Performing metrics to collect data for future analysis.

BAROQUE does not alter any of the collected data. All collected data is as if it was collected without BAROQUE. BAROQUE's sole purpose is to simplify the collection process and assist in scripting the quantum experiments. This saves time, allowing for more experiments to be performed and evaluated.

Chapter 5

Results

BAROQUE was used to gather the metrics and benchmarks identified in Section 3 by simulating IBM Manila, Quito, Oslo, Guadalupe, Toronto, and Washington. The 5-qubit and 7-qubit devices were simulated using an 11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz 2.30 GHz processor and 8GB of random-access memory (RAM). The higher qubit devices were simulated using RIT's Research Computing High Performance Computing Cluster [15].

5.1 Topology

The topology tests involved running circuits on a specific IBM topology at full occupancy. Then, each metric is evaluated for each test case to determine which algorithm(s) are best for that specific case. The best performing algorithms for each metric are then listed into a table for analysis. The tests results for the Δ CNOT and Δ depth metrics are compiled into Table 5.1 below.

Table 5.1: Topology Quantum Experiment Results - Gates and Depth

Topology	Circuit	# Qubits	Δ CNOT	Δ Depth
5-	QFT	5	SABRE/LKHD	SABRE/LKHD/STCH
5T	QFT	5	SABRE/LKHD	SABRE/LKHD/STCH
7H	QFT	7	SABRE/LKHD	STCH
16X	QFT	16	SABRE	SABRE/STCH
27X	QFT	27	SABRE	SABRE
5-	DJ	5	STCH	STCH
5T	DJ	5	SABRE/LKHD	SABRE/LKHD
7H	DJ	7	STCH	STCH
16X	DJ	16	STCH	STCH
27X	DJ	27	SABRE	SABRE
5-	WS	5	Negligible	Negligible
5T	WS	5	BSC/LKHD/STCH	BSC/LKHD/STCH
7H	WS	7	BSC/LKHD/STCH	BSC/LKHD/STCH
16X	WS	16	SABRE/LKHD	LKHD
27X	WS	27	SABRE	LKHD
5-	GHZ	5	Negligible	Negligible
5T	GHZ	5	SABRE/LKHD/STCH	SABRE/LKHD/STCH
7H	GHZ	7	SABRE/LKHD	SABRE/LKHD
16X	GHZ	16	SABRE	LKHD
27X	GHZ	27	SABRE	SABRE/LKHD

Table 5.1 shows the experimental results of the topology-based experiments. For each experiment, the routing algorithm(s) with the lowest Δ values are listed. If routing algorithms are very close or equivalent, they are listed together. In general, basic and HERR add more gates and layers than their more advanced counterparts:

SABRE, lookahead, and stochastic. There are some outliers in simple, small qubit circuits like GHZ and WS where the selected routing algorithm has no major effect on the gate count and depth of the routed circuit. Examining the complex circuits (DJ and QFT), two winning algorithms emerge. SABRE is consistently producing the least number of CNOT gates, while stochastic consistently reduces the total number of layer in the topology experiments.

Table 5.2: Topology Quantum Experiment Results - Accuracy and Timing

Topology	Circuit	# Qubits	Accuracy	Time
5-	QFT	5	-	SABRE
5T	QFT	5	-	SABRE
7H	QFT	7	-	SABRE
16X	QFT	16	-	SABRE
27X	QFT	27	-	SABRE
5-	DJ	5	SABRE/LKHD	SABRE
5T	DJ	5	SABRE/LKHD	SABRE
7H	DJ	7	SABRE/LKHD	SABRE
16X	DJ	16	STCH	SABRE
27X	DJ	27	-	SABRE
5-	WS	5	Negligible	SABRE
5T	WS	5	Negligible	SABRE
7H	WS	7	BSC/LKHD/SABRE	SABRE
16X	WS	16	SABRE	Negligible*
27X	WS	27	-	Negligible*
5-	GHZ	5	Negligible	SABRE
5T	GHZ	5	SABRE/LKHD/STCH	SABRE
7H	GHZ	7	SABRE/LKHD	SABRE
16X	GHZ	16	SABRE/STCH	Negligible*
27X	GHZ	27	-	Negligible*

SABRE dominates the transpilation time category. For QFT and DJ, SABRE is faster than all other algorithms. For WS and GHZ, the other algorithms eventually catch up, as there is not much complexity in these circuits. Lookahead swap experiences heavy exponential growth in transpilation time, to the point at which it is orders of magnitude slower than basic swap. Other algorithms are only slightly slower than SABRE, however. Tests with a result “Negligible*” denote a test where all algorithms but lookahead were similar, but lookahead was significantly slower.

Tests with a “-” were not performed with the accuracy metric. The more advanced algorithms, SABRE, stochastic, and lookahead, tend to outperform basic and HERR in the accuracy metric, although it is often close for WS and GHZ, even negligible at times. Topology seems to minimally affect accuracy past 5-qubit devices, as the high noise has much more impact on the accuracy than the shape of the coupling map.

5.2 Occupancy

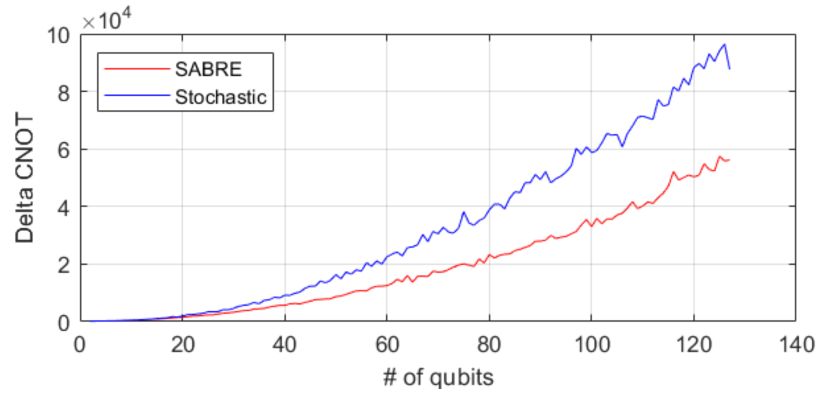
Occupancy is also important when determining which routing algorithm to select. There is no rule restricting the number of logical qubits to the number of available physical qubits on a device. Thus, circuits of each encoding type were selected and performed on high qubit devices with varying occupancy levels. For each test case, the best performing routing algorithm according to a specific metric was selected. The tests results for the Δ CNOT and Δ depth metrics are compiled into Table 5.3 below.

Table 5.3: Occupancy Quantum Experiment Results - Gates and Depth

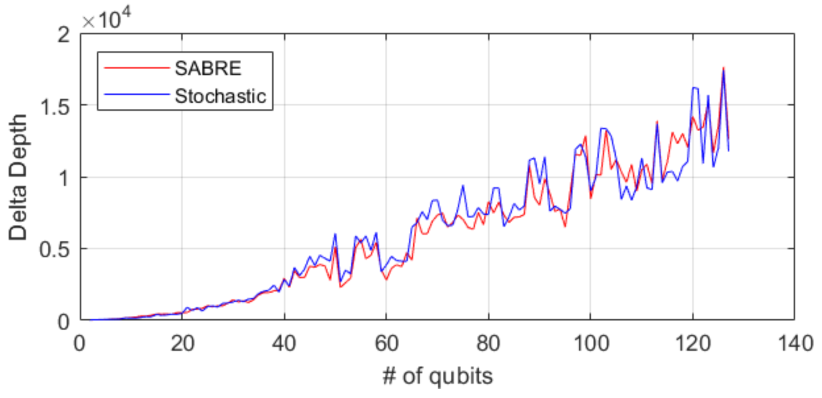
Topology	Circuit	# Qubits	Δ CNOT	Δ Depth
16X	QFT	4	SABRE/LKHD/STCH	STCH
16X	QFT	8	SABRE	STCH
16X	QFT	12	SABRE/LKHD	STCH
16X	QFT	16	SABRE/STCH	STCH
27X	QFT	7	SABRE/LKHD/STCH	SABRE/LKHD/STCH
27X	QFT	14	SABRE	STCH
27X	QFT	21	STCH	STCH
27X	QFT	27	SABRE	STCH
16X	DJ	4	SABRE/LKHD/STCH	SABRE/LKHD/STCH
16X	DJ	8	SABRE	STCH
16X	DJ	12	STCH	STCH
16X	DJ	16	SABRE	SABRE
27X	DJ	7	SABRE	STCH
27X	DJ	14	STCH	STCH
27X	DJ	21	SABRE	STCH
27X	DJ	27	SABRE	SABRE

Table 5.3 shows the results for the occupancy quantum experiments. The routing algorithms that produced the best Δ values are listed for each experiment. Two noticeable trends occur that match the results from the topology results. Based on Table 5.3, SABRE results in a lower CNOT count compared to other algorithms, while stochastic results in a lower circuit depth. However, the relatively small size of the Guadalupe and Toronto architectures does not paint the full picture. Thus, the top two competitor algorithms routed QFTs onto IBM Washington, one of IBM's 127-qubit processors. Figure 5.1 shows the CNOT and depth data obtained from

simulating QFTs onto IBM Washington.



(a) CNOT - SABRE v. Stochastic

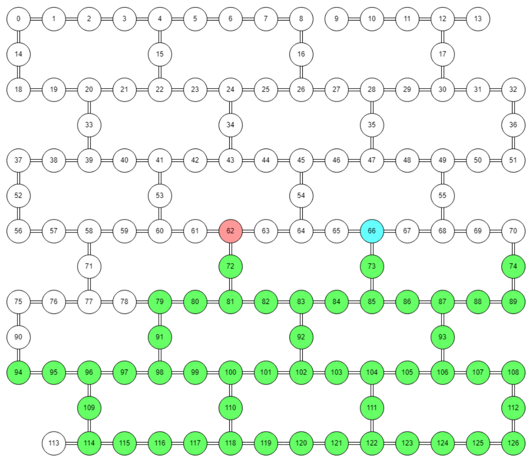


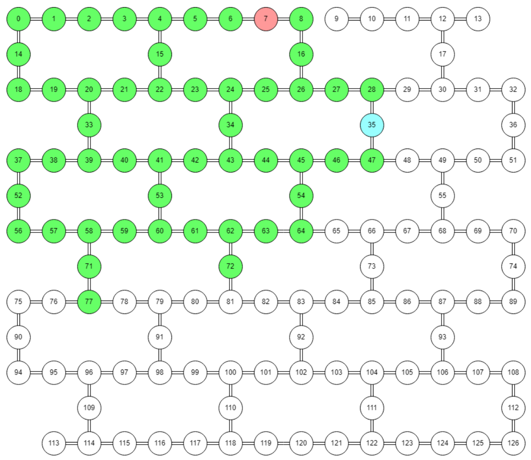
(b) Depth - SABRE v. Stochastic

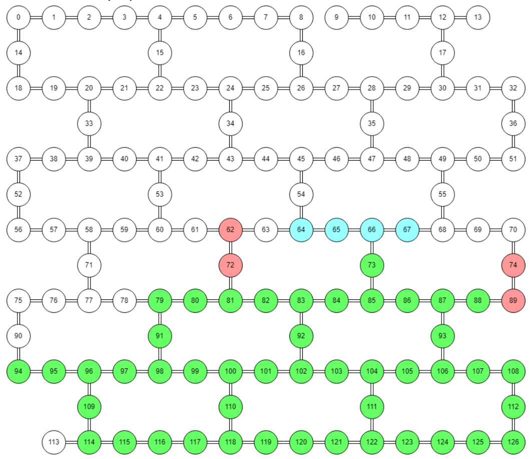
Figure 5.1: SABRE v. Stochastic - Routing 2-qubit to 127-qubit QFTs onto IBM Washington. SABRE's Δ CNOT scales better than stochastic's, but Δ depth is a close race.

Figure 5.1 provides crucial insight into the performance of these routing algorithms. SABRE produces less CNOT gates than stochastic does. This is a product of SABRE itself, as topology, occupancy, and circuit size do not seem to affect SABRE's dominant CNOT reduction, especially as circuit size increases, furthering the gap between itself and stochastic. Both algorithms show an upward trend in Δ CNOT as the number of qubits increases. However, there is variance qubit to qubit, as the CNOT gatesfor an n-qubit circuit are not necessarily greater than that for an $n-1$-qubit circuit. This causes a small spiking effect that is noticable in both SABRE's and stochastic's data. This effect is amplified by the Figure 5.1b. While SABRE was the clear winner in CNOT count, Δ depth is not as clear. SABRE and stochastic both trend upward in depth as the number of qubits increases, but not always. In fact, there are many places where depth will significantly decrease as even a single qubit is added to the circuit. This effect is likely topology and occupancy dependent, not routing algorithm dependent, as both SABRE and stochastic experience spikes at the same qubit counts, albeit at various degrees of severity.

It is not immediately clear what causes the large fluctuations in Δ depth. Since both SABRE and stochastic experience similar spiking patters, it is possible that the coupling map itself is the cause of this phenomenon. It is unlikely that all of the spike patterns are coincidences. To test this theory, a specific spike (50-qubits to 51-qubits) was examined by comparing the routed circuits. The following features were mapped for each routed circuit: initially mapped qubits that contained data at measurement (green), initially mapped qubits that did not contain data at measurement (red), and ancilla qubits that contained data at measurement (blue). White qubits are ancilla qubits that were not in the initial or final layout.


(a) 50-qubit QFT - SABRE


(b) 51-qubit QFT - SABRE


(c) 50-qubit QFT - Stochastic

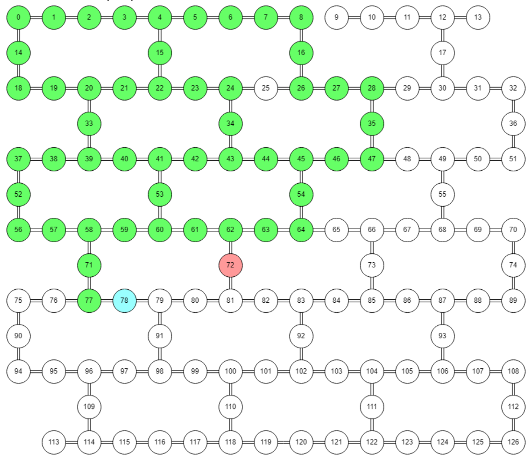
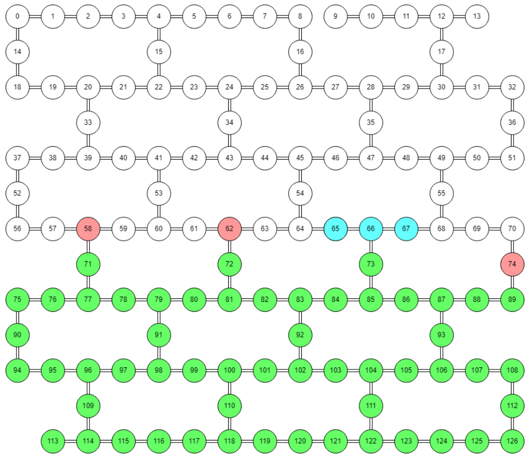

(d) 51-qubit QFT - Stochastic

Figure 5.2: Examining depth behavior: QFTs routed with SABRE and stochastic onto IBM Washington. There are noticeable patterns and similarities between SABRE and stochastic for both 50- and 51-qubit QFTs. The addition of another qubit shifts where the algorithms route and what qubits they utilize.

Figure 5.2 shows four results for routing a QFT onto IBM Washington. Notice the striking similarity in the qubit choices for SABRE and stochastic withing initial qubit mappings and the final mappings. Both algorithms decide to route in the bottom portion of Washington with the 50-qubit QFT, but route on the top left of the device for a 51-qubit QFT. The 50-qubit QFT utilizes five hexagon structures, while the 51-qubit QFT utilizes six. This is increased hexagon usage is a valid starting hypothesis as to why the depth for both routing algorithms drastically decreases from 50 qubits to 51 qubits. To determine if this hexagon hypothesis is correct, SABRE routed a 58-qubit and 59-qubit QFT onto Washington, shown in Figure 5.3.


(a) 58-qubit QFT - SABRE

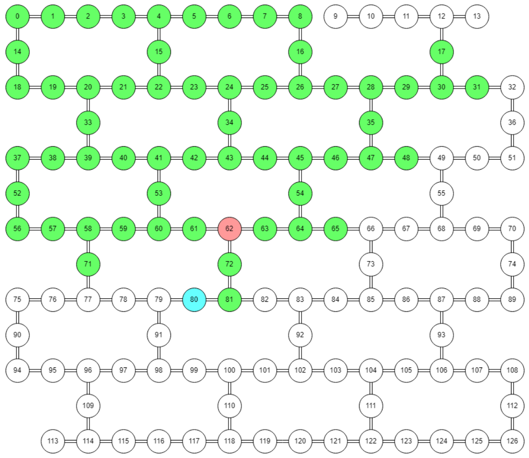

(b) 59-qubit QFT - SABRE

Figure 5.3: Examining depth behavior: More QFTs routed with SABRE onto IBM Washington. Similar patterns occur here; the addition of another qubit shifts SABRE's routing to a different portion of the hardware.

Both SABRE and stochastic experience another drastic drop in depth from 58 qubits to 59 qubits. However, Figure 5.3 shows that the number of fully utilized hexagons is six for both size circuits. Thus, the hexagon hypothesis is not entirely correct. The underlying reason is much more complex. Some series of qubits must be more efficient to route in certain configurations onto the quantum hardware, causing the mapped qubits to switch places from the bottom to the top left, as seen in these experiments. By changing where routing occurs, depth can be optimized due to the number of available adjacent qubits and the connectivity of that section. Thus, there must exist some quantum circuits of size n that are more efficient to route than a circuit sized $n-1$, as seen in Figure 5.1b. Therefore, there must exist some combination of properties in the target hardware that become utilized by an n-sized circuit that cannot be utilized by an $n-1$-sized circuit, such as a full hexagon, a qubit with high connectivity between hexagons, etc. From this, it can be inferred that a higher connectivity topology would result in a depth decrease compared to a similarly sized topology with lower connectivity. This also means that routing algorithms will avoid sections with low connectivity or high distance from the bulk of the utilized qubits (such as qubits 9 & 10 in the broken top right hexagon on IBM Washington). This information can be used to further improve routing and also design topologies with routing in mind.

SABRE dominated the CNOT category on Washington, outperforming stochastic on 98.5% of the QFT circuits. Depth was much closer, with SABRE only outperforming stochastic on 57.1% of the QFT circuits. This result is intriguing. Stochastic inserts many more CNOT gates, double the number introduced by SABRE with larger circuits, yet stochastic rivals SABRE in depth increase. To further explore this oddity, a new metric was produce. The *layer density* is a theoretical measurement of the total number of added CNOT gates divided by the total number of added layers. Layer density measures the average number of new CNOT gates per new layers. Note that layer density is not the actual average of gates per layer, instead its a theoretical measurement to estimate how many layers an algorithm will add to support the required CNOT gates for routing. Figure 5.4 plots the layer density for stochastic and SABRE for the IBM Washington QFT experiments.

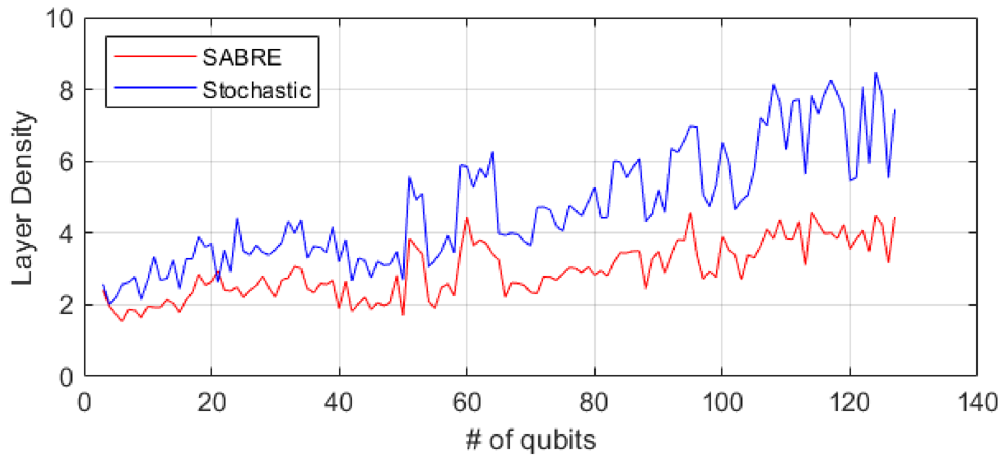


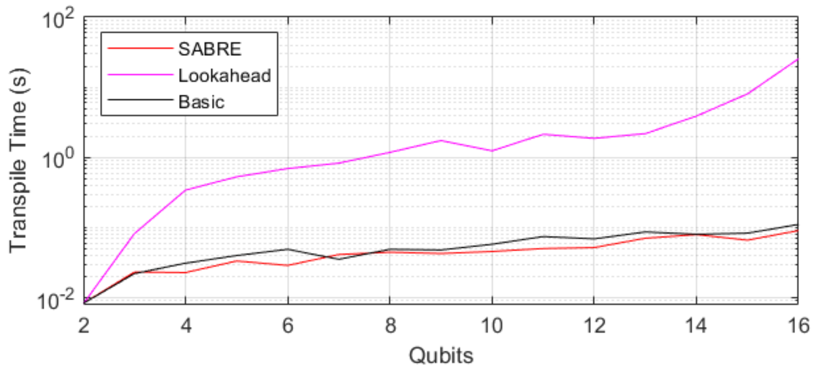
Figure 5.4: Layer density for SABRE and Stochastic routing QFTs onto IBM Washington. Stochastic swap packs CNOTs into layers more efficiently than SABRE does, resulting in stochastic staying competitive in Δ depth despite the high Δ CNOT.

The layer density measurement provides useful insights into how the algorithms work. First, the spiking in the data is due to the spiking of the depth and CNOT data, although it is primarily based on the depth due to the depth's high variance qubit-to-qubit. Second, stochastic layer density increases faster than SABRE. If a larger IBM machine was available to test, it can be inferred from Figure 5.4 that stochastic would continue to be competitive in depth count as the number of CNOT gates increase past SABRE. This is crucial, as it can support stochastic's viability as a useful routing algorithm for future quantum hardware. This also means that stochastic is more efficient at implementing same-layer SWAPs compared to SABRE, which is useful for overall execution time of quantum circuits.

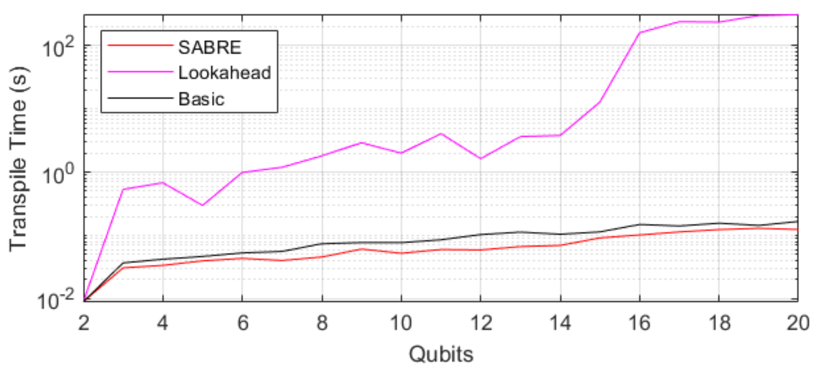
Table 5.4: Occupancy Quantum Experiment Results - Accuracy

Topology	Circuit	# Qubits	Accuracy
16X	DJ	4	SABRE
16X	DJ	8	SABRE
16X	DJ	12	SABRE
16X	DJ	16	STCH
27X	DJ	7	HERR
27X	DJ	14	SABRE/STCH

Table 5.4 shows the results of the accuracy for the DJ occupancy tests. Due to resource limitations, the 27X topology was not simulated with 21-qubits and 27-qubits. Ignoring one outlier, stochastic and SABRE produce the highest accuracy for a given experiment. Transpilation times are not listed, as HERR, basic, SABRE, and stochastic had negligible differences in their transpilation time. However, in all experiments, lookahead swap was significantly slower. Figure 5.5 shows timing data for lookahead swap compared to SABRE and basic swap.



(a) IBM Guadalupe - Deutsch-Jozsa transpile times



(b) IBM Toronto

Figure 5.5: IBM Toronto - Deutsch-Jozsa transpile times. Lookahead swap scales significantly worse than all other algorithms, quickly becoming orders of magnitude slower.

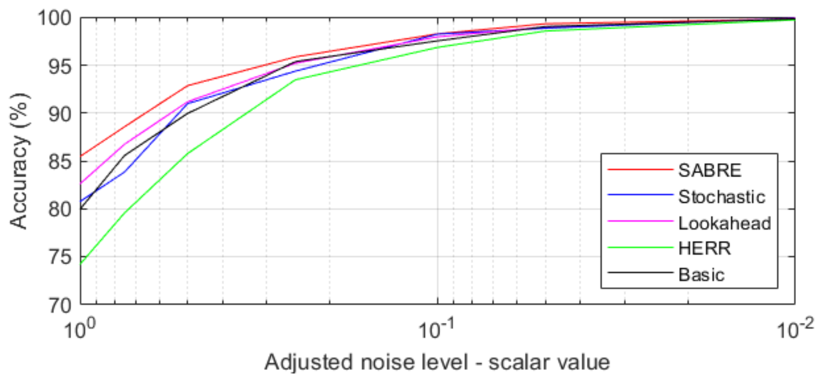
By transpiling routed circuits onto Gaudalupe and Toronto, the impact of routing algorithm on circuit transpilation time can be visualized. In both graphs, it is clear that as the size of the DJ circuit increases, lookahead becomes orders of magnitude slower than basic swap and SABRE swap. Note that HERR and stochastic have similar time tends to basic and SABRE, and are thus left out of the graph for readability. At full occupancy (16 qubits) on Guadalupe, lookahead is over 100x slower than the competitor algorithms. At 20 qubits on Toronto, lookahead is over 1000x slower. While lookahead operates in the minutes at this scale, this exponential increase essentially prevent lookahead swap from competing with the other algorithms due to the vastly inferior time complexity of its routing process.

5.3 Noise Level

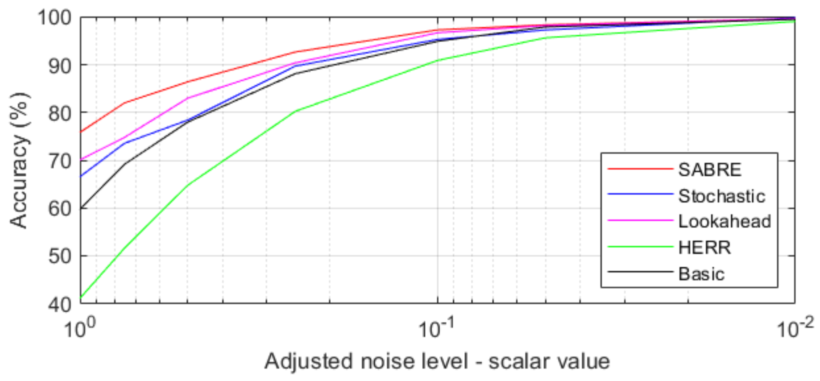
Noise level affects the accuracy of the evolved quantum state. Examine how routing algorithms handle various noise topologies will uncover which algorithms will produce more accurate results than other algorithms.

5.3.1 Adjusted Noise Level

The adjusted noise level experiments were performed on altered quantum hardware. The IBM Gaudalupe error map was adjusted by applying a scalar multiplier to every link in the map. All five routing algorithms were performed for both an 8-qubit and 12-qubit DJ circuit, with the noise scalar changing every time. Figure 5.6 shows the experiment results.



(a) 8-qubit DJ



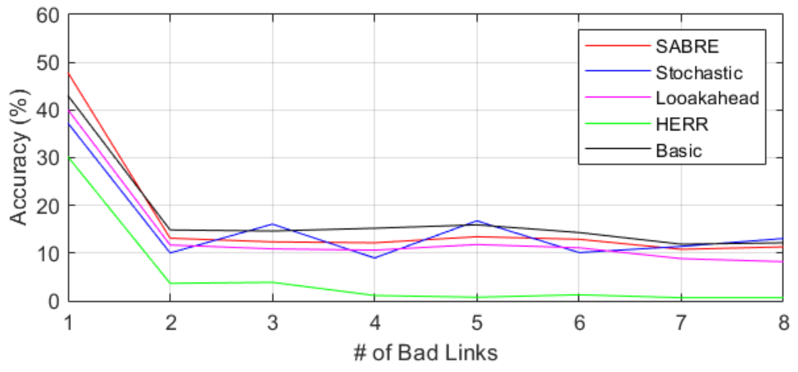
(b) 12-qubit DJ

Figure 5.6: Adjusted noise level experiment results. At nominal noise levels, certain routing algorithms out perform others. However, as the noise level decreases, the accuracies converge towards 100%, minimizing the accuracy advantage certain algorithms have.

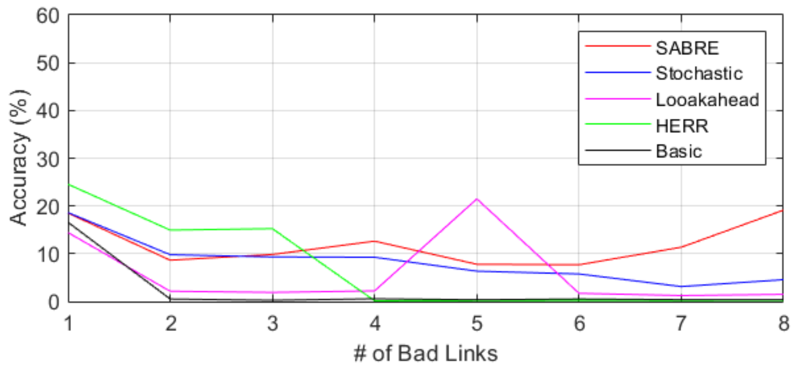
Figure 5.6 shows the accuracy trends of the routing algorithms as the noise map is scaled. There are two notable trends here. First, complex routing algorithms outperform basic swap and HERR in terms of accuracy. Second, and most important, as noise decreases, the accuracy of each routed circuit approaches 100%. This inference is crucial as it shows that reducing the noise of quantum computers will lower the routing algorithm's impact on the accuracy of the circuit, thus reducing the need for routing algorithms to specifically target accuracy optimizations.

5.3.2 Bad Links

The bad links experiment alters the noise of specific links by greatly increasing their noise level. This data can show how a routing algorithm is affected by certain links introducing irregularly high error into the quantum state. Furthermore, this experiment can be used to test the feasibility of noise aware routing algorithms, such as HERR. All experiments were performed on the IBM Toronto architecture.



(a) 12-qubit DJ on IBM Toronto



(b) 16-qubit DJ on IBM Toronto

Figure 5.7: The bad links experiment results show that the introduction of a bad link may or may not impact the accuracy of a routed circuit, and that noise-aware algorithms have the potential to avoid bad links.

Figure 5.7 reveals helpful trends and limitations for routing algorithms. The nominal noise map of the quantum computers already introduces significant noise when performing 12-qubit and 16-qubit DJ circuits. Due to this, a single SWAP performed on a bad link will not significantly alter the quantum state. However, performing multiple SWAP gates on one or more bad links can introduce enough noise to significantly alter the quantum state. This phenomenon can be seen in Figure 5.7a, specifically in the SABRE data, as well as the large dip in accuracy with two bad links. The accuracy of the routed circuit is not always altered by the introduction of another bad link. SABRE's accuracy does not always dip when another bad link is introduced. If a bad link is not utilized by the routed circuit, then its noise level does not affect the circuit's outcome. However, sometimes a bad link is placed in a critical part of the circuit. For example, the first bad link is introduced near the edge of the coupling map. The second bad link is placed at a central intersection between Toronto's two hexagons. Another takeaway is that HERR is the most accurate algorithm for the first three bad link tests in Figure 5.7b

5.4 Review of Routing Algorithm Performances and Test Conditions

The data collected in the quantum experiments is useful for grading the usefully of algorithms in general, as well as aid in the identification of target use cases for said algorithms.

5.4.1 Lookahead Swap

Lookahead swap produces accuracies, Δ CNOT, and Δ depth that rival stochastic swap and SABRE on small qubit circuits. Despite this, lookahead is not feasible for large circuit routing. By generating many possible layouts for each possible SWAP, lookahead's routing and transpilation times become orders of magnitude worse than stochastic and SABRE. Even worse, lookahead does not improve upon the other algorithms enough to justify the high routing time. If lookahead was leagues above

SABRE and stochastic, it could be worth the time. Unfortunately, this is not the case. The inferior time complexity of lookahead swap restricts its feasibility to small qubit circuits.

5.4.2 Basic Swap & HERR

Basic swap and HERR are not detrimental to the routing process and they offer little against stochastic and SABRE, save for a few niche cases. Primarily, small qubit circuits are simple enough that the other routing algorithm's strengths have little impact. For example, the difference between SABRE and basic routing a 5-qubit QFT exists, but is minimal. However, since stochastic and SABRE tend to outperform basic and HERR in CNOT count, depth, accuracy, and timing, there is minimal reason to select basic and HERR over stochastic and SABRE in most cases.

Basic and HERR are not entirely useless, nor outclassed by stochastic and SABRE. Basic swap and HERR work just as well as stochastic and SABRE on simple, small circuits, such as WS, GHZ, or any circuit with minimal routing required. Basic swap is a simple algorithm, making it perfect for educating students on the basics of quantum routing. HERR is relatively weak as a noise-aware routing algorithm due to its heavy reliance on basic swap. HERR suffers the same shortcomings as basic. Furthermore, the noisy nature of current hardware means that HERR often fails to find suitable noise-aware routes. However, HERR is useful as a stepping stone to more complex noise-aware routing algorithms. Perhaps HERR's methods could be adapted to stochastic and SABRE to work on future less-noisy hardware. HERR is limited by the limited connectivity of IBM's hexagon architecture. A topology with higher qubit connectivity could improve the usefulness of noise-aware routing algorithms like HERR. As a bright spot, HERR is proven to be effective over other algorithms when a bad link disturbs the accuracies of non-noise-aware algorithms.

5.4.3 Stochastic Swap & SABRE

Stochastic and SABRE frequently outperformed their competitor algorithms in the vast majority of the quantum experiments. SABRE is often slightly faster than stochastic, but not by much. However, SABRE and stochastic fill different niches from each other. SABRE aims to balance CNOT count with routing time. As such, SABRE is both relatively fast at routing, and consistently produces the lowest CNOT counts compared to its competitors. The data corroborates this, as SABRE leads in Δ CNOT and accuracy metrics. Stochastic swap tries to find the best SWAP configuration per each layer. By restricting the algorithm to one layer at a time, stochastic can miss a more optimal solution that spans multiple layers. This is reflected in the data, as stochastic and SABRE are very close when it comes to depth, yet stochastic is not nearly as effective at reducing CNOT count. It is also worth noting that stochastic struggles with very simple circuits, such as GHZ and WS, compared to SABRE, while this is not true for DJ and QFT.

The data from the quantum experiments shows the two categories of metrics the algorithms target. SABRE successfully reduces CNOT count while increasing accuracy. Stochastic decreases the number of additional layers introduced to the circuit through routing due to its high layer density when routing. If reducing CNOTs and producing the highest possible accuracy, SABRE is the safe bet. For reducing the number of layers required to apply a circuit, either SABRE or stochastic will produce the best result. Since both are very fast algorithms running both and checking their depth count is a within the realm of feasibility for large circuits on large quantum processors.

Chapter 6

Conclusion

6.1 Future of Routing Algorithms

The future of routing algorithms will be guided by the improvement of the size and quality of quantum hardware. As quantum computers increase in qubit count, so will quantum circuits. The increased size will have many affects on routing algorithm selection. First, slower algorithms, such as lookahead swap, will become obsolete as their effect on transpilation time grows quickly with increasing scales. Second, routing will become more complex, resulting in both an increased CNOT count and routed circuit depth. Thus, routing algorithms that reduce gate count and or circuit depth will become the default.

As quantum computing technology improves, and as the NISQ era ends, quantum noise will become less of an issue. In the event this occurs, accuracy becomes less of a concern when performing quantum circuits on hardware. Furthermore, as more qubits are introduced, some algorithms may require more shots in order to build the final probability distribution. The combination of lower noise levels with more complex circuits will mean that circuit depth will become the key metric for routing algorithms to target. Less depth requires less time to execute, an effect that compounds as more shots are required. In the event that two algorithms produce equivalent accuracy and transpilation time, but one algorithm results in a shallower circuit, the routing algorithm that reduces the circuit depth will be preferred.

6.2 Future Work

This work opens four paths for future related work. Expanding the test sets into higher qubit topologies can confirm, alter, or uncover tends that will aid in the development of the future's quantum routing algorithms. IBM's quantum computers follow a similar architecture. Performing tests on non-hexagon based topologies, such as Google's, can uncover more use cases for the routing algorithms other than SABRE and stochastic. Testing larger topologies of varying connectivity can alter which algorithms are better. Second, identifying current routing algorithms that are promising for routing future hardware is crucial, as those algorithms, such as SABRE and stochastic swap, can be improved to route more complex circuit on larger quantum hardware. Creating alternate version of SABRE and stochastic that specifically aim to reduce circuit depth can start the next chapter in the development of quantum routing algorithms. Third, combining SABRE and stochastic into a routing algorithm capable of high layer density with less CNOT count than stochastic may prove to be an effect algorithm for future hardware. Finally, examining the exact cause of the depth spikes in Figure 5.1b and the mapping behavior in Figures 5.2 and 5.3 can uncover features that, when implemented, improve the efficiency of the topology or open avenues for routing algorithms targeting those advantageous features.

6.3 Conclusion

SABRE and Stochastic Swap are two of the best quantum routing algorithms available. SABRE is fast and effective at reducing CNOT count, this increasing the accuracy of the routed circuit. Stochastic has much higher layer density than SABRE, staying competitive in depth reduction despite the much higher CNOT gate count.

The future of quantum computing will look for lower circuit depth to reduce circuit execution times. SABRE and stochastic are stepping stones to improved algorithms that target depth and advantageous hardware features. For now, SABRE and stochastic outclass their competitor algorithms.

Bibliography

[1] IBM, "Ibm quantum compute resources," 2022, accessed October 2022. [Online]. Available: https://quantum-computing.ibm.com/services/resources?tab=systems

[2] Qiskit Development Team, "Open-Source Quantum Development," https://qiskit.org.

[3] ——, "Basicswap," 2021, accessed October 2022. [Online]. Available: https://qiskit.org/documentation/stubs/qiskit.transpiler.passes.BasicSwap.html#qiskit.transpiler.passes.BasicSwap

[4] S. Jandura, "Improving a quantum compiler," 2018, accessed October 2022. [Online]. Available: https://medium.com/qiskit/improving-a-quantum-compiler-48410d7a7084

[5] Qiskit Development Team, "Stochasticswap," 2021, accessed October 2022. [Online]. Available: https://qiskit.org/documentation/stubs/qiskit.transpiler.passes.StochasticSwap.html#qiskit.transpiler.passes.StochasticSwap

[6] G. Nannicini *et al.*, "Optimal qubit assignment and routing via integer programming." 2021, accessed October 2022. [Online]. Available: https://arxiv.org/abs/2106.06446

[7] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for nisq-era quantum devices," 2019, accessed October 2022. [Online]. Available: https://arxiv.org/abs/1809.02573

[8] Qiskit Development Team, "Sabreswap," 2021, accessed October 2022. [Online]. Available: https://qiskit.org/documentation/stubs/qiskit.transpiler.passes.SabreSwap.html

[9] S. Bonaventure, "High error rate qubit routing," 2022.

[10] A. Zulehner, A. Paler, and R. Wille, "An efficient methodology for mapping quantum circuits to the ibm qx architectures," 2018, accessed November 2022. [Online]. Available: https://arxiv.org/pdf/1712.04722.pdf

[11] C. C. Aggarwal, A. Hinnebyrg, and D. A. Keim, "On the surprising behavior of distance metrics in high dimensional space," 2001.

[12] L. K. Grover, "A fast quantum mechanical algorithm for database search," *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing - STOC 96*, 1996.

[13] A. W. Harrow, A. Hassidim, and S. Lloyd, "Quantum algorithm for linear systems of equations," *Phys. Rev. Lett.*, vol. 103, p. 150502, 2009.

[14] N. Quetschlich, L. Burgholzer, and R. Wille, “MQT Bench: Benchmarking software and design automation tools for quantum computing,” 2022, MQT Bench is available at https://www.cda.cit.tum.de/mqtbench/.

[15] Rochester Institute of Technology, “Research computing services,” 2022. [Online]. Available: https://www.rit.edu/researchcomputing/

[16] IBM Quantum, “IBM Quantum Composer,” https://quantum-computing.ibm.com/composer/docs/iqx.

[17] IBMQuantum, “IBM Quantum Lab,” https://quantum-computing.ibm.com/lab/docs/iql/.

[18] Qiskit Development Team, “qiskit.circuit.QuantumCircuit.count_ops,” https://qiskit.org/documentation/stubs/qiskit.circuit.Quantum-Circuit.count_ops.htm.

[19] G. Li, Y. Ding, and Y. Xie, “Tackling the Qubit Mapping Problem for NISQ-Era Quantum Devices,” May 2019, arXiv:1809.02573 [quant-ph]. [Online]. Available: http://arxiv.org/abs/1809.02573

[20] S. Jandura, “Improving a Quantum Compiler ,” https://medium.com/qiskit/improving-a-quantum-compiler-48410d7a7084, 2018.