University of Sheffield

Automated Design Simplification of Quantum Program Synthesis

George O'Brien

Supervisor: John Clark

A report submitted in partial fulfilment of the requirements
for the degree of PhD in Computer Science

in the

Department of Computer Science

24th September 2021

Declaration

All sentences or passages quoted in this document from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations that are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure.

Name: _____

Signature: _____

Date: _____

Acknowledgements

I would like to thank everyone who provided support to the publication of this work. From academic staff guiding my writing and research, to friends who listened to me talk about nonsense over lunch. With my great fortune, the full list is much too large to be contained here.

I'd like to bring special attention to all of my various supervisors, especially John Clark, for their guidance on what ended up being a much more complicated research topic than planned.

In addition, I'd like to thank both Sheffield's Verification and Testing office as well as the unofficial Quantum Computing group for being generous with their time and expertise to help wherever they could.

Publications

Papers

George O'Brien and John A. Clark. "Using Genetic Improvement to Retarget quantum Software on Differing Hardware." 2021 IEEE/ACM International Workshop on Genetic Improvement (GI). IEEE, 2021.

Works in Progress

George O'Brien and John A. Clark. *Simplifying Synthesis of Quantum Programs Using Automated Scheduling.*

George O'Brien and John A. Clark. *Automating the Building of Quantum Transpilation Pipelines with Predictive Classifiers.*

Abstract

Quantum computers are a new and emerging technology that offer promises of being able to outperform classical machines. However, they differ from classical machines so much that they provide unique challenges to development. Working on quantum machines is currently very difficult, requiring a large amount of expertise in a great deal of areas. In order to facilitate practical software engineering methods it will be necessary to greatly simplify this process.
To provide this process simplification we identify automation methods and approaches that can perform steps of quantum program compilation to greatly reduce the need for human expertise.

The first contribution looks at integrating an existing classical method into the quantum model. This is done through the application of a Genetic Improvement algorithm. The second contribution looks at modelling the quantum compilation problem in a way compatible with a classical model. This is done through the generation of a Planning Domain Definition Language (PDDL) model. The third and final contribution looks at simplifying the building of a compilation stack. This is done by using a neural network to make decisions about what steps to add to the compilation stack.

The results of this show a set of automated methods that produce error rates competitive with the standard quantum compilation methods. In addition, these methods require much less expertise about specific quantum hardware or the quantum compilation stack and are built to be compatible with the current IBM Quantum Experience software stack.

Contents

List of Figures

Chapter 1

Introduction

1.1 Quantum Computing at a Glance

Historically, computing problems have been dominated by standard transistor-based computer models. As improvements have been made, the size of transistors has shrunk year after year allowing monumental increases in power. However, these improvements are increasingly becoming unsustainable as transistor size reaches its physical minimum [75]. If increased performance and problem-solving capacity are desired, then it is necessary to look at alternative approaches.

Quantum computing is one of the most promising alternative approaches to classical computing. The initial theory can be traced to Paul Benioff's [7] proposal of a quantum implementation of a Turing machine. This was built on by separately Richard Feynman [34] and Yuri Manin [55], who proposed that a quantum machine could do things that a classical machine could not. This was theoretically demonstrated with Peter Shor's factorisation algorithm [76], which can efficiently solve factorization problems in polynomial time.

This theoretic backing has provided the promise of quantum computer superiority being applied to real problems. While factorization [77] has been mentioned, there are other areas such as particle simulation [20] or approximate optimization [30] which can provide real value to industrial fields.

Until very recently, quantum hardware has not been sufficient to practically implement these methods. However, the last few years have produced a turning point with what is often described as a new "era" of quantum computation. These are referred to as Near-term Intermediate Scale Quantum (NISQ) computers [70]. These provide barely sufficient computing power to test realistic and practical quantum programs in a way that was previously impossible.

It is currently very difficult to implement quantum algorithms that can provide measurable advantages to industry software work. Developing quantum algorithms is, simply put, very difficult. Standard software engineering tools built into the classical algorithm development process are either very new or do not exist at all. This makes it necessary for algorithm designers to have a detailed understanding of the system in order to fully exploit the hardware. As quantum computing is moving from a specialist theory-driven field into one with a practical software component, the problems caused by the difficulty have been largely neglected.

This means that there is currently great utility in simplifying this process where possible. By lowering the barrier to entry it becomes easier, faster and less expensive to attract contributors to the field. In academic areas, this can mean a greater ability to perform interdisciplinary work. In commercial settings, this can mean reducing the resource costs of recruitment and development.

The current approach to quantum computing with the best software engineering structure is IBM's gate-based quantum machines, the IBM-Q. These provide a set of quantum development tools that can be used to work on quantum algorithms one layer at a time. While these are functional for a full theory-to-implementation development cycle, many steps require human expertise. It is possible in many of these areas to remove the human requirement through automation. However, the method of automation is not straightforward as almost all stages require decisions to be made that require an insight into the quantum hardware. Any automation process must therefore take advantage of the current state of the art of automated analysis and decision making and carefully apply it to selected steps.

The following thesis aims to take a more practical, software-focused view of quantum computing algorithm design with the aim to build simpler approaches to minimizing error on real quantum machines. We look at quantum computing algorithm design as an iterative set of modular steps that translates a high-level abstraction to a real physical machine. We therefore research methods that allow the automation of these steps without requiring explicit knowledge of the top-level abstraction or the low-level hardware.

The practical immediate benefit of this work is providing example proofs of concept to automation methods and providing example tools that have been integrated into real software stacks. The long-term benefit of this research is helping to build a stronger software-engineering foundation for the development of quantum computing and taking steps to enable a greater variety of non-quantum expertise to contribute to the field.

1.2 Research Gaps

A More Practical View of Quantum

Quantum computing approaches are focused on theory and proof of concept. The main paradigms of quantum computing, Grover Operators, Quantum Fourier Transforms, Harrow-Hassidim-Lloyd algorithms, Variational Quantum Eigenvalue solvers, and direct Hamiltonian simulations. During the literature survey it was noted that the vast majority of papers included these in the motivation but did not conclude with a practical description of how one could use their paper's research to implement or improve the motivating algorithm in a useful way. Instead, it was presented as sufficient that if the motivating algorithm was useful and that an arbitrary paper's research applied to it. This is at odds with what would be seen in wider computer science - where directly useful software is often expected. It is likely that this approach is in part caused by the fact that quantum machines did not allow direct implementations, even toy implementations. However, the latest quantum machines mean more practical approaches are possible and example implementations are much more reasonable to expect.

Quantum Computers are Not Theoretical Objects

It is currently difficult to incorporate real quantum machine information into quantum approaches. Not only are there multiple kinds of quantum hardware (circuit-based, annealling, hybrid) that are radically different from each other, even within those models there is large difference. Publically available circuit-based models have thoroughly measured values for these properties, including average error of individual gates and qubits. However, the sheer complexity of this information means that it rarely appears in modern research papers. Instead, it is often treated with basic but useful assumptions (for example "lower average error is better"). These assumptions save sufficient time over hand choosing but do provide inaccuracies. Developing methods of automatically considering this can remove these inaccuracies without requiring users to spend time hand picking algorithms.

Quantum Software Engineering is Underdeveloped

Classical software engineering is far ahead of quantum software engineering. Classical models of testing, language design and function abstraction has been developed for decades. Quantum computing theory is a much newer subject and simply does not have this ground work. However, with modern quantum languages being developed (such as qiskit) it is important to look at classical software engineering principles before attempting to develop new methods. In particular, we look at the flexible, modular approach that classical function design uses. Trying to emulate this with quantum circuit design is important to create software that can be applied to a variety of machines and problems.

Quantum Development is Still Too Hard

Quantum software development is currently very hard, relying on a wide range of knowledge of very specialist subjects. Mainline quantum software like qasm needs to hand-allocate gates to qubits to build an algorithm for a specific machine. Even assuming a background in both the quantum particles involved in the hardware and the algorithm's computer science theory, there are a great many details of the error rates, qubit arrangement and implementable gate sets. These properties change how a user would build an algorithm in ways that are non-trivial and non-obvious. There is a gap in approaches designed for simplification of the process, not just performance!

1.2.1 Addressing the Gaps

The main method of addressing these gaps is looking at automated methods of developing quantum systems. The chapters in this thesis look at genetic improvement, AI planning and AI classification. We adapt these systems to model real quantum machines and take into account the real limitations and error rates of the hardware. We also look at designing these as modular steps in a larger step-wise structure, helping to build a stronger software stack.

1.3 Purpose of the Work

The purpose of this work is to provide researched examples of automated and structured methods of building quantum code on real quantum hardware. The advantage of automation that we aim to exploit is the increased speed and ease of automatic development. We also look at providing more structured methods to allow new innovations to be more easily integrated with a more standardized quantum pipeline. We finally look at "real" quantum computers, meaning hardware models that take their properties from the specifications of machines that exist. This allows a focus on methods that can be practically tested and exploring how quantum theory can take to a practical environment. This brings the overall purpose to show that practical, potentially heuristic methods can be applied to provide better solutions to current quantum computing problems.

1.3.1 Research Aims

Our summary aims are:

- Provide proof of concept of automated methods of building quantum software
- Integrating these methods into practical software stacks to improve the range of models they can be applied to while also reducing the required expertise to do so

- Ensure that these methods are compatible with the real quantum systems, being able to both automatically integrate hardware and build off state-of-the-art software

1.4 Overview of the document

1.4.1 Chapters 2 and 3: Background Literature

In these two chapters, we provide an overview of the literature that the research is built on. This starts with an introduction to quantum computing, including the hardware, the software, uses as well as why it is thought to have huge potential. After this, we look at some practical quantum computers, with a focus on the IBM-Q machines which see considerable use as practical models throughout this thesis. After this, we move on to the problem of the program and circuit synthesis (the main problem addressed in this work). We then discuss software stack engineering as a method of performing this synthesis. Finally, a summary of the gaps in the literature is discussed as well as promising approaches to addressing them.

1.4.2 Chapter 4: Initial Probe and Research Aims

In this chapter, we describe the research motivation behind the chosen topics. We address and justify the three main objectives of our research. These are: automating the quantum code generation, providing structure to quantum algorithm design, and integrating our approaches with real quantum hardware.

The topic of this thesis has changed considerably since its original proposal. The initial proposal focused on looking at how to apply a step-wise verification approach to algorithm design. However, the early reading into the area showed that what the process required most was methods that are simple and have softer constraints.

1.4.3 Chapter 5: Automated Circuit Design

Chapter 5 looks at applying a genetic improvement method to the problem of quantum circuit synthesis. Our approach takes a graph-based population of quantum circuits and applies iterated modifications that aim to minimize error. This targets real quantum hardware. The key result is generated using a python implementation of the method. **The result shows an overall reduction in error** when generating circuits using this method as opposed to a standard one. The results are particularly interesting for two reasons. First, the method only requires a standard circuit and hardware model input, both of which are accessible and is therefore both a simple and effective method to implementing quantum code. Secondly, genetic improvement has had a long theoretical justification for application in quantum computing. By providing a practical example of this, we can bring more weight to the value of the theoretic work.

These results were accepted for publication at The 10th International Workshop on Genetic Improvement, ICSE 2021 ([64]).

1.4.4 Chapter 6: Automated Scheduling of Quantum Bits

Chapter 6 Results

For the second contribution we look at building a planning model to solve quantum scheduling problems. Planning problems already have a strong software development structure meaning that if compatible models are provided a wide range of tools can be made compatible. The key result is the generated schedules that provide instructions for a quantum circuit. Once again the **results shows an overall reduction in error**. In addition, the results provide an interesting structure that integrates the quantum circuit problem into a planning domain model. This allows non-quantum experts to build planning AI technology to target a quantum problem with a lower knowledge barrier to entry.

1.4.5 Chapter 7: Automated Software Stack Construction

For the third contribution, we look at providing automation directly to the software stack itself. This was implemented by building an automated classifier to suggest the best optimization methods to build a software stack. The resulting program takes in gate information from a circuit and returns a selection of pre-written methodologies it predicts are likely to provide the most improvement. The results generated were an accuracy assessment of the classifiers ability to predict whether a method would be effective. The **results show that the method can predict the best option with sufficient accuracy for the tool to be useful** (upwards of 80% in the best case and significantly better than random chance in almost all cases). This method is interesting as it does not aim to replace hand-optimization but work alongside it. This helps automate expert decision making by providing initial structures without requiring expertise. This helps simplify the design process by lowering the workload of experts and also extending the range of work that partial-experts can perform.

1.4.6 Chapter 8: Conclusion

In the final chapter, we discuss the findings of this thesis. We highlight the main contributions and what they might mean in the wider field. We discuss how these results look in relation to our original thesis aims. We also discuss what limitations our results have and what needs to be considered before making conclusions on them. We suggest some areas of future work that are promising to build upon.

Chapter 2

Literature Survey: Quantum Computing

2.1 Overview of Literature

2.2 Introduction

Quantum Computers are an entirely unique paradigm of computation based on quantum mechanical phenomena. [62] [83] Quantum computers are built up from their core building block, called quantum bits or qubits. Qubits perform an analogous role to a classical bit. However, where a classical bit can be encoded as either a 1 or a 0, a quantum bit can be put into additional combination states [60]. These combination states are called superpositions. While the theoretical properties[1] of a superposition mean it must be seen as a unique quantum combination, it is useful to use a model of classical distributions to represent it. In this case, a superposition [71] is represented by a probability distribution which corresponds to the likelihood of the qubit returning 1 or 0 when measurement in relation to a specific basis. We can perform transformative operations [73] on these distributions in much the same way as we can perform operations on a regular bit. However, as these are being applied to the entire distribution, we can perform certain operations much faster. In fact, it is thought that quantum computers can work in an entire complexity class above classical computers.

Quantum computers are showing themselves capable of outperforming classical computers when given certain difficult problems, such as factorization [77], particle simulation [20] or approximate optimization [30]. The promise of quantum supremacy has led to huge interest in development in the field. However, there is no definitive answer as to which method will provide the best method to demonstrate a quantum approach that can outperform a classical one. This has meant that many different lan-

[1] This paragraph aims to be a useful simplified summary. Further detail and accuracy are included later in this chapter.

guages [25] [39] [11] [93], models [14] [42] [2] [46] [68] and algorithms [19] [96] [80] are being developed simultaneously. In order to be able to take advantage of the state-of-the-art, effort must be made to ensure that these methods can be mixed and matched at different stages. This is a complicated process, as even the easiest methods require some hand-tuning. To be adapted to new machinery or compilation approaches, algorithms must be looked at again from a top level. This is at odds with the modern software engineering process which greatly emphasise using methodologies that allow a more iterable and modulated design process. Overall, this makes quantum circuit development very hard and require a high degree of expertise.

One of the major focuses of this paper is quantum research that lowers this level of required expertise. Therefore, we start with an overview of quantum computing aimed at non-experts. This is to give an understanding of what quantum computers are and what makes them different. In particular, we aim to provide quantum circuits in terms of an "observable" representation. An observable is a matrix representation aimed to capture the observable behaviour of a qubit. Observables are one of the simpler methods of representation and are intuitive to anyone who has worked with matrices. Further on in the work, we look deeper into this. There we focus on the technical specifics including more accurate mathematical models, the basis in quantum physics and what behaviour it entails, and the specific details and background information of the more common software and hardware.

In this section of the literature survey, we provide an introduction to quantum computing. We provide a context of the quantum state-of-the-art and explain why modern quantum computing is at such a pivotal moment. We demonstrate some example technologies to implement quantum machines. We further analyze these methods with a focus on their software engineering properties. We then highlight some weaknesses in these designs that cause an increase in complexity of development.

2.2.1 Quantum bits

The basic unit of quantum computing, called a quantum bit or qubit, is the key to the advantage that quantum computers provide. A single qubit represents a single quantum state and is implemented with the quantum properties of a physical quantum system. For example, a qubit might be represented by an electron's spin characteristic. A qubit has the same potential states as a classical bit, either 0 or 1 [60]. However, a qubit can also exist in a uniquely quantum combination of these two states known as a superposition [71]. This gives them a state described by the following equation:

$$| \phi\rangle = \alpha\bullet \, | \begin{bmatrix}1\\0\end{bmatrix}\rangle + \beta\bullet \, | \begin{bmatrix}0\\1\end{bmatrix}\rangle$$

with α and β being complex numbers equal to the square roots of the respective probabilities *amplitudes*. These can, equally, be described as a single two by two

matrix in the form

$$| \phi\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

When observed the state will only ever return 0 or 1 with a fixed probability, despite being the combination of two states [13]. The square of the magnitude of the coefficient amplitudes $p(0) =\mid \alpha \mid^2$ and $p(1) =\mid \beta \mid^2$ provides the likelihood of a one or a zero. These probabilities must sum to exactly one. Why super-positions act this way is not known and is very far out of scope.

2.2.2 State Representation

This means that a single qubit can be conveniently represented as a vector. Furthermore, all possible states of a single qubit can be represented as a point on a sphere. This leads directly into how one would represent a quantum gate. Each quantum gate represents a single unitary transformation matrix that acts upon the point in the sphere that represents the input qubit [58]. The state of an n-qubit system is a 2^{n} dimensional vector of length 1. For single-qubit gates, these are two by two matrices to represent the real and complex position. We change the state of a qubit with these rotations, moving it from one point on the surface to another without ever reducing its length. Gates acting on more than one qubit work in a similar way, with the rotations affecting the combined dimensional state space [**gates**].

To help visualize the state of a single qubit, it is possible to represent the state space as a point on the surface of a sphere. This is called the Bloch sphere [97]. The top and bottom of the sphere correspond to the classical equivalent states of $\mid 0\rangle$ and $\mid 1\rangle$, with any complex vector that can be expressed as a rotation of these being a valid state. The length of the state's amplitude vector equals the radius and remains constant.

2.2.3 Quantum gates

In practice, while any unitary operation is theoretically viable, quantum computing hardware is only compatible with a finite set of operations [67]. This finite set varies based on which quantum computer one is using as each one represents a physical action the computer can take.

These quantum states can then be manipulated using quantum gates. This is analogous to the way classical gates use Boolean operators, with quantum gates instead of using *unitary transformations* on the quantum state. Unitary transforms are a powerful set of operations that have very useful properties, such as being physically reversible. A unitary transformation is often represented by a $2^n \times 2^n$ unitary matrix, whose entries are complex numbers and where n is the number of qubits. A unitary

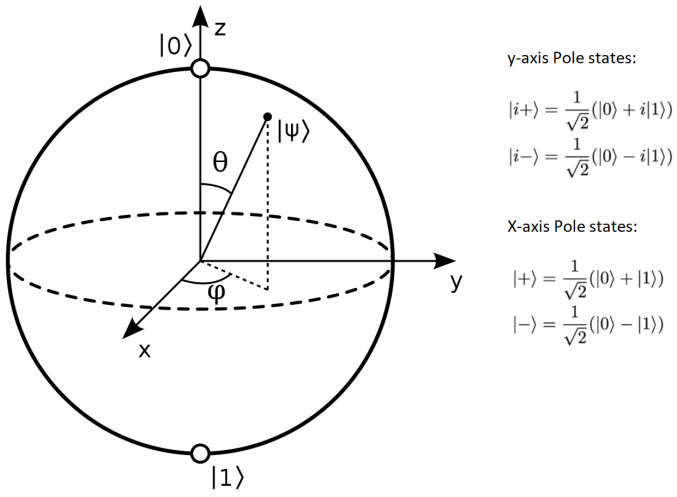


Figure 2.1: *A diagram of a Bloch sphere with the non-classical pole positions explicitly described*

transformation's matrix U satisfies the equation $UU^\dagger = U^\dagger U = I$, where † denotes conjugate transpose (i.e. the rows and columns have been swapped with each other as have the real and imaginary components). Thus, the conjugate transpose of U is its inverse.

Multi-qubit gates

Multi-qubit gates can be represented using the same generalized definition as a single qubit gate ($2^n \times 2^n$ with $n > 1$) [61]. However, introducing multiple qubits to a single operation adds further complexity than a single qubit. This is because of quantum entanglement. Entangled states are ones where a measurement on one has implications on the states of the other. This means that a combined entangled qubit state can no longer be represented as two separate qubits, only as a single combined state.

The most famous example of this is a Bell state [78]. When a Bell state is measured, 50% of the time it will return $|\ 11\rangle$ and 50% of the time it will return $|\ 00\rangle$. This also means there is a 0% chance of either $|\ 01\rangle$ or $|\ 10\rangle$. If each qubit is looked at individually, it can be seen that each has a 50% chance of returning a 1 or 0. If these are naively combined, it would show a 25% chance of each of the four possible states. This is a different distribution to the actual Bell pair and represents a comparative loss of data!

Exploiting entanglement [23] to build more data into distributions is one of the sources of quantum advantage [72].

2.2.4 Examples of Quantum Gates

Quantum Gate Composition Example: Quantum Not-gate

As our first example, we look at the implementation of a classical Not gate as a quantum gate. A classical Not takes an input of 0 and returns 1 or takes an input of 1 and returns 0. To apply a not gate to a qubit, we can start by considering the classically analogous states of $\mid 1\rangle$ and $\mid 0\rangle$. For any state of exactly $\mid 1\rangle$, we would expect a quantum Not-gate to change the state to $\mid 0\rangle$. That is to say, when measured, we would expect a result that would have returned a 0 to instead return a 1. This means that a quantum Not gate should flip the amplitude vector of the qubit in relation to the $\mid 1\rangle$ / $\mid 0\rangle$ plane. We note that this plane is the X-axis, making a quantum Not-gate a 180° rotation around the X-axis. This is also called a Pauli-X gate. Pauli-Y and Pauli-Z gates also exist, referring to similar rotations around the Y- and Z-axis respectively. We represent a Pauli-X gate as the following unitary matrix:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Properties of Quantum Gate Example: Hadamard Gate

We now look at an example of a quantum exclusive quantum gate. This is the Hadamard Gate. A Hadamard gate applies to a single qubit and maps a state of either $\mid 1\rangle$ or $\mid 0\rangle$ to $\frac{|1\rangle - |0\rangle}{\sqrt{2}}$ and $\frac{|1\rangle + |0\rangle}{\sqrt{2}}$ respectively. The output states both map to identical probabilities on measurement - an equal chance of returning either 1 or 0. This is represented by the following unitary matrix:

$$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

This can equally be represented as a 90^{o} rotation around the Y-axis, followed by a 180^{o} rotation around the X-axis. The Hadamard represents a core property of a qubit: the ability to form a superposition. In addition, the function outputs returned from an input of $\mid 1\rangle$ and $\mid 0\rangle$ are two different states, despite being equivalent when measured. When considering how many classical bits would be required to represent these same properties, the relative power of the qubit becomes clear. In addition, consider the amount of classical processing required to apply the equivalent of even a single Hadamard gate!

Two Qubit Quantum Gate Example: Expanding the Hadamard

The core of two-qubit gates is the same as single-qubit gates. Each gate is described by a unitary matrix describing rotations across the collective amplitude vector. However,

as you increase the number of qubits you increase the dimensional of the representation space. This can make diagrams and intuitional understanding difficult. We, therefore, demonstrate an expansion of the Hadamard function to show how it would be applied to two qubits at once. This will leave both qubits in the same 50/50 superposition as the single-qubit Hadamard. This is equivalent to the application of a discrete Fourier transform.

The Hadamard function H is defined recursively. For any value of m the appropriate function H_m can be calculated as such:

$$H_m = \frac{1}{\sqrt{2}} \begin{bmatrix} H_{m-1} & H_{m-1} \\ H_{m-1} & -H_{m-1} \end{bmatrix}$$

Where $H_1 = 1$.

Therefore, H_2 can be defined as:

$$H_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} & \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \\ \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} & -\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \end{bmatrix}$$

$$H_2 = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ 1 & -1 & -1 & -1 \end{bmatrix}$$

This shows the structure of a two-qubit gate as a $4x4$ ($2nx2n$ where $n = 2$) matrix. In addition, while the Hadamard function does represent a simple expansion, it shows the difficulty of expanding a quantum gate. With a classical logic gate, each bit is independent meaning scaling is relatively simple. With the quantum gates, combined qubit operations must account for entanglement and the connected nature of qubits. This means that simultaneous applications to qubits create much more complicated structures.

2.2.5 ZX: Representing Quantum Circuits Graphically

We introduce ZX [22] [94] as a graphical method of representing quantum circuits. ZX provides a language that allows generalized Z and X operations extending the range of what can be represented. This allows users to describe and reason with quantum circuits while maintaining the soundness. This usefully allows access to generalizable rewrite rules, which are much harder to implement on fixed quantum circuits. Generalizable rewrite rules are useful in implementing certain automated search techniques (such as Genetic Algorithms) and therefore ZX provides a unique value to the research.

```
1   OPENQASM 2.0;
2   include "qelib1.inc";
3
4   qreg q[5];
5   creg c[5];
6
7   h q[0];
8   cx q[0],q[1];
9   measure q[1] -> c[1];
10  measure q[0] -> c[0];
```

Figure 2.2: *A simple Qasm program represente diagrammaticly. We see a Hadamard gate preceding a controlled not gate.*

The core of ZX graphs are nodes called spiders. Each spider can contain an number m of input wires and n output wires which represent $(C2)^{\otimes m} \rightarrow (C2)^{\otimes n}$. The spiders apply one of two functions. Green spiders indicate the function $\mid 0...0 >< 0...0 \mid + e^{i\alpha} \mid 1...1 >< 1...1 \mid$ while red spiders indicate the function $\mid +...+ >< +...+ \mid + e^{i\alpha} \mid -...- >< -...- \mid$. In simple terms, the green spider represents an arbitrary matrix transformation. The red spider is similar but differs in that it indicates that the state must be put in the Hadamard basis first through the application of quantum Fourier transforms (called Hadamard functions). This basis is undone when re-entering a green node and the state is returned to the standard computational basis.

In practice, this means that green spiders represent the computational basis while red spiders are represented in the Hadamard basis. That is to say the green spiders are some combination of computational vectors, while red spiders are superpositional vectors. $(\frac{1}{\sqrt{2}}1 0, \frac{1}{\sqrt{2}}0 1)$ $(\{\frac{1}{\sqrt{2}}1 1, \frac{1}{\sqrt{2}}1 -1\})$

We then show a simple example program, and redraw it as ZX Graph:

---

h q[0]; cx q[0],q[1];

This then translates to the following graphical representation:

We present an equivalent ZX:

We can see the similarity to the original code. The Hadamard gate is now represented by a blue line. The CNOT is now two connected nodes, with the control as a green node and the target node as a red node.

We also identify a subset of ZX-graphs called circuit-like graphs. These graphs can be more easily converted back to circuits. This subset is defined by the following four rules:

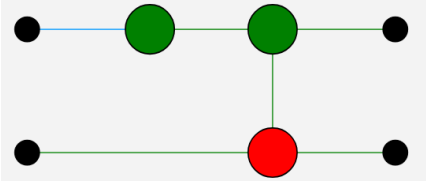


Figure 2.3: *A simple example of a ZX-graph, demonstrating a Hadamard gate as a blue line and two connected spiders forming a controlled not structure.*

- All spiders are Z-spiders.
- Z-spiders are only connected via Hadamard edges.
- There are no parallel Hadamard edges or self-loops.
- Every input or output is connected to a Z-spider and every Z-spider is connected to at most one input or output.

It is proven that any ZX-graph can be represented as a circuit-like graph.

2.2.6 Quantum "Advantage"

While the theoretical potential of quantum computers would surpass classical machines with ease, current models are still imperfect. The current era of machines is referred to as Noisy Intermediate Scale Quantum computer (NISQ) [70]. Noisy refers to the quality of the qubits, which is lower than would be expected for a pure Quantum model. Intermediate Scale refers to the size of the computers themselves, in particular, a range between 50-100 qubits. However, these size ranges are only guidelines. Quantum Computers below 50 qubits have been developed with NISQ frameworks with the idea that any work done on these can be scaled up when NISQ computers are more accessible. Both the noise and the size means we are limited in the scope of the problem we can look at. The noise limits the number of gates that can be used in a single circuit before combined error becomes too large. While the qubit number provides obvious limitations in computational power. However, NISQ computers have been shown to be able to outperform certain classical computers.

Physical Implementation

A quantum computer with an n-qubit storage register is implemented physically from n qubits [11]. Unlike a classical n-bit register which can only be placed in one of the

2^n basic n-bit states, an n-qubit register can be placed in in a *complex superposition* of all 2^n basis states $\mid 000...0 >$ through $\mid 111...1 >$. The state amplitude vector of the superposition $\mid \phi\rangle$ is represented by a 2^n by 1 column vector of complex numbers:

$$\mid \phi\rangle = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \\ \alpha_{2^n-1} \end{bmatrix}$$

where

$$\Sigma_{i=0}^{2^n-1} \mid \alpha_i \mid^2 = 1$$

Measurements on the entire state will cause collapse of the state to $\mid i\rangle$ with probability $\mid \alpha_i \mid^2$. Unitary transformations essentially apply a rotation over the 2^n-dimensional state space. As before, any single state may correspond to one of the 2^n basis states or a complex superposition. Applying a gate to a superposition results in the superposition of the application of that gate to each of the basis states. Therefore, for a gate implementing a unitary transformation U we have:

$$U(\Sigma_{i=0}^{2^n-1}\alpha_i \mid i >= \Sigma_{i=0}^{2^n-1}\alpha_i U \mid i >)$$

Which, for a physically implemented machine, has the additional requirement that the gate must implement an action that the computer is capable of physically performing.

2.2.7 Other Archetypes of Quantum Computing

Quantum Automata and Quantum Turing Machines

The original quantum computers were theoretic models. The original of these was Benioff's 1980 [8] construction of a quantum mechanical model of a Turing machine. While this was capable of existing in quantum states between computation steps, it was required to return to a classical state at the end of each step. A later model, introduced by Deutsch in 1985 [29], takes a different approach that allows the tape to exist in quantum states. While these are not new models, they are still entirely relevant to the modern theory.

A few practical applications of quantum automata have been found such as direct integration with interactive proof programs [63]. However, it seems that it is not possible to use them directly in quantum programming languages. This does limit their direct relevance to any practical quantum software research.

Measurement-based

Measurement-based or one-way quantum computing [15] is done by computing an entangled quantum state then performing a single qubit measurement on it that destroys

the state. The destruction of the state is the source of the "one-way" property of the approach. The ability to reason in terms of state preparation and measurement provide potential advantages over using only circuit-based models. A full calculus description of measurement-based quantum [27] has been developed that allows reasoning about measurement-based quantum programs.

There's the promise that measurement could play an even more involved role in quantum computing. Proposals have been made showing that quantum teleportation can be exploited to provide unique approaches to quantum calculation [38]. If fully developed, a teleporting gate model would allow the implementation of a universal quantum computer with only projective measurement, quantum memory and preparation of the $\mid 0\rangle$. However, further work is required to make this practical.

Topological

Topological quantum computing [59] and topological quantum error correction [99] employ similar theories to build more stable quantum computers. For topological quantum computers, the model traditionally uses two-dimensional particles called anyons to construct entwined "braids" that can be used to implement logic gates. Small changes would not change the topological properties of these braids, making the system much more resistant to quantum decoherence [65]. Topological error correction uses similar entwining approaches but instead uses these to implement error-correcting codes more efficiently by creating topological patterns that can be checked. This allows the error correction to be implemented with fewer additional qubits.

Adiabatic

Adiabatic quantum computing [2] is one of the more uniquely quantum approaches, without anything in the way of a classical equivalent. Adiabatic quantum computing uses a non-discrete (i.e. a continuous) model of time when computing. This uses a physical implementation of a Hamiltonian that slowly varies. The initial state is implemented as the Hamiltonian's ground state with the solution provided by the final state. Provided the system is evolved slowly enough, it is possible to guarantee that the final state of the system will only differ from the ground state of the final Hamiltonian by a negligible amount. This means the solution can be obtained by measurement with a high probability. This is a very promising model of quantum computation. However, it's much more difficult to develop using classically-analogous software techniques that rely on an assumption of discrete operations.

2.2.8 Quantum Error Correction

Quantum Error Correction [53] is a technique used to protect a quantum computer from errors. NISQ-era quantum machines are noisy, with error coming in from a number

of areas such as gate-induced error, measurement error and noise from decoherence. If a quantum computers error rate can be brought beneath a certain threshold, the logical error rate can be brought to arbitrarily low levels. This is referred to as the quantum threshold theorem [3]. Quantum computing beyond this point is referred to as fault-resistant quantum computing. Reasonably efficiently implemented fault-resistant quantum computers is expected to provide a huge expansion to the scope of what can be computed. Many proposals have been made describing how it would be able to outperform classical machines at certain problems.

Basics of Error Correction

The simplest method of classical error correction uses multiple copies of stored data to allow redundancy in case of error [100]. If a single copy has an error, the redundant parts can be used to correct it. However, quantum states cannot be copied as described by the no-cloning theorem. This means that the error correction methods must be more complicated [16]. This usually means spreading the quantum state across multiple highly-entangled qubits to build redundancy. This is similar, in principle, to the classical method. However, implementing it is much more complex. In addition, the resource cost is much higher even before consideration of how much higher the cost of a qubit is compared to a classical bit.

How Far Away is Quantum Error Correction

While QEC offers a lot of potential power to quantum machines it is still a long way from being practical [17]. Experiments have been successfully performed showing that QEC is real and can work [24]. However, the resource cost is too high to provide practical machinery. While it's impossible to discuss quantum computing without mentioning QEC as an end goal, it is difficult to argue in favour of assumptions that fault tolerance is likely to be imminent. Therefore, when planning research, we instead intend to focus on taking advantage of the currently available resources and minimizing the current limitations. Even if not directly applicable to the exact designs of future quantum machines, exploring the principles and theory of quantum error methods provides useful understanding and is worthy of exploration.

2.2.9 Quantum Error Mitigation

As discussed, error correction provides a method of fault-tolerant quantum computing. However, quantum error correction requires a larger number of qubits than is possible with current hardware. It is therefore worth looking at similar currently applicable methods that can provide similar functionality. One of these kinds of methods is error mitigation.

Basics of Error Mitigation

Error mitigation [33] is an alternative method to provide a level of fault tolerance. While it cannot provide the same benefits as fully developed error-correcting codes, in the short term it can provide efficient improvements to quantum computing. Error mitigation is applied by identifying areas of well-characterized error and applying their inverse to the circuit. These error-mitigated circuits return solutions closer to a theoretical error-free circuit.

The main problem in applying this approach is generating a well-characterized error distribution. In order to do this, individual sources of error must first be identified. This allows sampling or analytical methods to be applied to smaller sections to generate individual error profiles.

Per Operation Error Correction

Our model of quantum computation includes a set of operators aimed at transforming the initial qubit states into the final measured states [33]. Each of these operators can be modelled as a combination of a theoretical error-free version of the operator (O^n) and an addition error distribution (e). Per operator, error correction consists of adjusting the observable final state to counteract the error distribution. For each of O^n, we can do this by applying a noise operator equal to e^{-1}. Alternatively, for any small value of e, a set of linearly independent basis operators can be applied to compensate it.

Hardware Bias Error Correction

In addition to operation error, quantum hardware systems themselves also introduce error [10]. Individual Qubits have a natural bias towards specific resting states that happens in a predictable fashion. This happens both as a result of applying operators as well as time passing. This error can be similarly modelled as an error distribution. This error distribution can be similarly inversed and applied as a corrective noise operator.

2.2.10 Context of the Quantum Computing Research Landscape

The Quantum Computing landscape has changed drastically in the last few years. While previously the focus has been on the theory and capabilities of quantum computers, the hardware has advanced to the stage where practical quantum advantage is looking imminent. This has driven on work in practical quantum software development with the aim of implementing quantum advantage. Papers have been collected to try and show the current landscape through this lens. Three papers have been selected based on their ability to reflect the wider academic landscape. One provides the context

of quantum computing theory, one provides a case for short-term quantum hardware and one looks at analyzing differences in modern machines.

QC paper 1: Foundational Theory of Quantum Computing

Andrew Steane's paper [81] is an early work in Quantum Computing. It describes the core differences in quantum and classical theory. It presents the foundation theory of Quantum Computing. This makes the case that quantum and classical information theory is different enough to require an entirely new field. The case starts with a summary of the classical theory of computation. It then branches into a discussion of quantum versus classical physics and the difference between the two. It identifies an example (The Bell-EPR argument) to demonstrate that a task that is physically possible but where no classical computer could perform it. In addition, it shows that with a quantum understanding of the problem the task is relatively simple.

It then goes on to describe modern (1997) quantum information theory. Further focus on several key properties of quantum information theory. It describes the theoretically ideal quantum computer, referred to as a Universal Quantum Computer. Notably, this is not a Universal Fault-tolerant Quantum Computer, the modern theoretically ideal Quantum Computer, as Quantum Error Correction is relatively new (formalized in 1996) at the time of publication. Instead, the focus is moved directly into a summary of quantum algorithms and hardware. Only after this is Quantum Error Correction described. Finally, the paper is summarized with a discussion that provides one of the stronger cases for the theory of Quantum Information Theory.

This paper is a survey of the physics-driven approach to Quantum Computing. It provides a comprehensive summary of the theory behind quantum computation. While the technology has moved forward, the fundamental theory and the level of interest physics researchers have in exploring this theory is just as true now as it was at the time of publication. This paper is therefore useful to the larger context of the thesis by providing a case that the theory of quantum computing is interesting to larger academia.

A case is presented for quantum information theory as an area analogous to classical information theory. It starts by affirming that any isolated quantum system can always be described with unitary operators. However, as no truly isolated system can exist, instead we approximate collections of quantum systems using the Schrodinger equation. A method of handling this approximation is to model a system and its environment. The Schrodinger equation measures the evolution of the system. At least one non-unitary evolution must be added between the system and its environment. This is a characterization of the measurement.

By taking Nature (or real systems) as essentially information processors, we can

therefore look at them as computations to be simulated. This is useful as a quantum bit is an exact description of a quantum system, containing a complete description with no extraneous information. This is obviously a useful property when attempting to simulate larger quantum systems.

It is useful to focus on a more concrete definition to draw similarities to classical computing. A quantum equivalent of the Church-Turing thesis is described based on a similar principle proposed by Deutsch (1985): *Every finitely realizable physical system can be simulated arbitrarily closely by a universal model computing machine operating by finite means.* This highlights the idea that it is not necessarily true that a universal Turing machine could compute all behaviour found in nature, while it may be physically possible to realise a new type of computation separate from classical computer science. Or, simply put, it is possible that quantum computing is a different complexity class to classical computing.

An example of a simple task that cannot be performed by classical computers is the EPR (Einstein-Podolski-Rosen) paradox [6]. This is an experiment involving pairs of two-state quantum systems. Each pair consists of an up and down spin particle. They are prepared in a singlet state and then fly apart in opposite directions across the y-axis. The question is whether this can be entirely characterized by quantum physics. While the initial analysis showed no difference in the behaviour of classical and quantum information approaches, a more thorough approach found a difference. When the spin component of the two particles along different axes in the x-z plane, each measurement yields an answer of + or -. However, there is no way to assign the properties of the different particles independently. This makes it classically impossible to implement a full classical simulation of the system despite it being clearly physically possible. This in turn demonstrates a quantum system that cannot be classically defined.

Quantum bits and processing provide unique properties that don't exist classically. Quantum Entanglement is considered a method of information encoding. The fact that we can transmit at least two classical bits worth of information in a single quantum bit allows the necessary justification that qubits have more storage capacity than a classical bit. In addition to this storage capacity, entanglement enables quantum teleportation. This enables the transfer of information without further transfer of physical qubits, only classical information. In this way, entanglement allows information encoding in messages as well as static data. While exploiting these properties is difficult, these demonstrate properties that are not classically available as well as methods that have an advantage over classical methods.

The important algorithms of quantum computing have remained much the same since their introduction. The two major algorithms are Shor's factorisation algorithm and Grover's search algorithm. In addition to these algorithms, there are more generic

fields that benefit from quantum approaches. An example of this useful to physics research is Quantum Simulation, mostly consisting of calculating Schrodinger equations. Similarly, Quantum Cryptography, specifically Quantum Key Distribution, can be used as an example with more clear use in commercial software. It is highlighted that while there are not many quantum algorithms, this is made up for with how powerful the available ones are.

This now provides sufficient context to generate a reasonable case for Quantum Computing research. The pillars of this argument are that there exist problems that cannot be done classically, some of these problems may be solvable quantumly and that it is feasible that a Quantum Computer can be built that would be capable of running these solutions. In addition, it is shown that there are uniquely Quantum properties that cannot be replicated classically and that these provide advantages in certain situations. Finally, examples are provided of specific problems that take advantage of specific Quantum properties. These examples can be shown to take advantage of Quantum properties, are expected to outperform classical equivalents and, most importantly, are of tangible practical use. This cements the case of Quantum computing as feasible, otherwise unachievable and practically useful.

QC paper 2: Modern Quantum Computing

John Preskill's paper [70] is a fundamental paper on Quantum Computing. It describes what is known as a Noisy Intermediate-Scale Quantum (NISQ) computer. It presents the state of modern (2018) Quantum Computing. The case starts with the potential of Quantum Computing which focuses mainly on its use for simulations. Specific examples of areas where quantum computing is likely to be useful are mentioned. It predicts that the more development Quantum Computing gets, the faster further development will be. It highlights quantum error correction as an eventual goal but suggests that in the short term its development will need to keep noise-minimization in mind. This is further expanded upon when discussing specifically hardware error. Finally, it concludes with the idea that quantum development should focus on paving the way for further more advanced systems down the road rather than immediate miracles. This paper has a great deal of popularity in both academic and industrial areas. It is often used to motivate an approach to Quantum Computing that focuses on setting strong developmental foundations. This is the same philosophy that this work uses. It is therefore a useful paper in understanding how the context of quantum research justifies this approach.

The attraction of Quantum Computing is there appears to be an area, referred to here as a frontier, that is only accessible with Quantum properties. This is referred to as the "complexity frontier" or "entanglement frontier". This refers to our new ability to build and control complex, entangled quantum states consisting of multiple particles. These states exceed our ability to simulate them with even the most powerful

classical computer. In addition, these cannot be characterized well with existing theoretical tools. From a physics perspective, the reason these machines are interesting is that there is strong evidence that they are capable of efficiently simulating any particle system. This, therefore, means they can simulate any natural system. The simulation of complex and esoteric particle systems allows an entirely new way of approaching fundamental physics.

The confidence that this will be beneficial relies on two areas. First is Quantum Complexity, which is the idea that Quantum Computers do solve problems in an entirely different complexity class. This allows them to efficiently solve problems entirely unsolvable classically. The second is Quantum Error Correction, which is the method expected to allow quantum computers to scale enough to solve hard and interesting problems. The core of both of these is Quantum Entanglement. In particular, it's the idea that the exploitation of Quantum Entanglement allows us to process large amounts of data simultaneously. This is the feature that provides an advantage to Quantum Machines. Classically describing a single additional qubit requires characterizing its entanglement properties with every single other qubit. Therefore, each linear addition of a qubit represented classically has a quadratic cost. This linear addition to representing quadratic amounts of classical data is the feature that makes Quantum Computing vastly different and, in many ways, more complicated than classical machines. Exploiting and understanding entanglement is a key step to achieving the true potential of quantum computing.

Three areas are highlighted as particularly notable evidence that quantum research is interesting. Firstly, we look at Quantum Algorithms for classically intractable problems. There are examples of problems that cannot be efficiently solved classically for which quantum algorithms exist. The best known of these is the algorithm for finding prime factors for large composite integers. While it is possible that these problems are solvable classically, the lack of success in the area indicates that it is not likely.

From a more abstract perspective, there are arguments using complexity theory proving that quantum states have superclassical properties. While these make assumptions that are not necessarily true, they are reasonable assumptions. An example is presented of sampling from a correlated probability distribution, a sampling that cannot be reproduced classically.

Finally, there is no classical algorithm that can simulate a quantum computer or other quantum systems. This is despite decades of effort from a wide array of different physicists to find better or more accessible ways. The amount of research in the area indicates that it is likely a classical algorithm cannot be made to simulate a quantum machine efficiently.

Overall this brings the conclusion that there exists a subset of problems that can

only be solved by Quantum computing. It is highlighted that these problems include simulations of particles in nature, which is a large and valuable area of study.

The main barricade of Quantum Computing is that it is very difficult to run and implement Quantum Programs. A fundamental feature of quantum programs is that you cannot observe them without disturbing the system. This leaves a position where the system must be perfectly isolated to maintain its state while also be externally controllable. This is a challenging task. Eventually it is expected that quantum error correction will be able to minimize this problem. However, in the meantime, the computational cost overhead of applying quantum error correction is too high making it unlikely to be implemented in the near future.

As the barriers for fault-tolerant quantum computing are still too great we turn to NISQ era quantum computers instead. While "noisy" and "quantum" are self-explanatory, the intermediate scale referred to by NISQ is less exact. The intermediate-scale refers to between fifty and a few hundred qubits. The lower bound is what's most important with 50 qubits representing the barrier between simulate-able by the largest classical supercomputer in existence. The upper bound exists more as a rough estimate of "large-scale" quantum computing, which is less distinct.

It is also important to consider the quality of the qubits in addition to their number. NISQ machines are noisy in ways that do not necessarily scale well with larger machines. Minimizing the noise error, through hardware or software, allows better scaling of the machines. This is essential in the NISQ era, where the quantum advantage is possible but quantum error-correcting does not exist.

We conclude with examples of promising problems for NISQ era quantum computers. The first described is approximation algorithms for NP-hard optimization problems. This is a well-known algorithm called QAOA, with a hybrid quantum-classical equivalent called a Variational Quantum Eigensolver. Secondly, quantum testbeds are described. These are heuristic approaches relying on practical tests rather than pure theory. There are many classical examples of the performance of these being high where the theory has not yet found an exact explanation. Quantum annealing is then described. The computational power of quantum annealing is not yet fully known, indicating further research is needed. Noise-resilient quantum circuits are described next. The focus here is on the idea that, without changing the error of a single gate, the clever design of programs allows the effect of the noise to be minimized. This includes software engineering and model-based approaches to quantum error, such as those proposed later in this paper.

These areas are extended by looking at areas that NISQ era quantum computers may be able to solve efficiently. These are Deep Learning, Matrix Inversion, recommendation systems, semidefinite programming and quantum simulation. While all of

these have heavy use in some or all of academic, industrial or commercial settings, Quantum Simulation is highlighted as of particular interest to the author.

This paper overall highlights the problems and opportunities in NISQ-era computing. This provides useful context to the thesis, which focuses on practical improvements in NISQ-era hardware. Of particular note is the idea that NISQ-era computing paves a path for error-resistant quantum computing. This helps justify building strong foundations at present. While those exact techniques may not have a lasting appeal, providing methodologies and structure to build upon is useful to the long-term development of the field.

QC paper 3: Quantum Hardware Relevancy

The final piece of context in modern quantum computing research is an understanding of the current state of the hardware. We, therefore, introduce Linke et al.'s paper [54] comparing two modern (2017) pieces of quantum hardware. This is useful context for two reasons. Firstly, it provides an insight into the capabilities of two quantum computers. At a very base level, these capabilities are slightly different indicating a trivial but necessary conclusion that quantum hardware influences performance in some way. Secondly, through comparison, it highlights the areas that are most relevant in making comparisons. This indicates that these are properties that are of interest to the authors and, based on the popularity of the paper, to the wider quantum world. A major topic of the larger thesis is considering and preserving properties, which makes knowing what properties are considered important by various quantum researchers is useful.

Two quantum computers are introduced. The first is a microwave resonator while the second is a linear chain of trapped ions connected by laser-mediated interactions. The first contains a single central qubit with a single connection to each of four other qubits, each of which has no other connections. The second consists of five qubits all connected to all other qubits.

Properties of the two quantum computers of interest are listed. These are the connectivity pattern, hardware type and gate library. In addition, there is further information about gate counts required for the implementations of important quantum gates and algorithms. These are the Margolus, Toffoli, Bernstein-Vazirani, Hidden shift, QFT-3 and QFT-5. As this was a paper that garnered interest there is an implication that these properties are interesting to a wider audience.

This paper highlights an interest in comparisons of different hardware in different quantum computers. It demonstrates that different quantum computers with different properties perform differently. It also demonstrates that these differences are dependent on what algorithms are being looked at. It also highlights various properties that were

decided to be interesting, with the main focus of these properties being the efficiency at which they implement algorithms. This provides a clear case that research into implementing quantum algorithms on different hardware has some interest in the wider research community.

2.3 Uses of Quantum Computing

Quantum computers are capable of creating solutions to certain problems with a better complexity class than Classical solutions. While identifying these problems can be difficult, there are common properties that can help. The major problem is ensuring that a problem can be represented efficiently in a quantum fashion. This is most efficient if the problem can be stored directly as a probability distribution. This can also be extended to matrix and vector space problems, which have a similar structure. In addition, Quantum systems are inherently probabilistic. This can mean some problems are made more difficult to solve, as they require deterministic results. However, this also means inherently probabilistic problems can be performed faster than classical ones. Examples of these types of problems are search problems, neural-network models and simulations.

2.3.1 Search Problems

Quantum computers perform exceptionally well in search problems [37]. To manage this each option of the search space is encoded in a single multi-qubit superposition. Then, a function is created that aims to check for the searched property. If it is found, an amplitude of one phase will be generated and if not the same amplitude in the other phase can be generated. If the search space superposition is formed in such a way that all values are in the same phase. This means any value without the given property will have its amplitude cancel out providing a new probability of zero. On the other hand, any value with the property will interact constructively, increasing its amplitude. Where applicable, this allows a lower complexity class. The main downside of these is that the current viable search algorithms are heavily constrained. For example, Grover's algorithm is a popular quantum search algorithm. It requires there to be no searchable structure in the problems, there to be an equal amount of inputs and possible solutions and there to exist a boolean function that can determine whether the answer is correct. While this is possible to work around, it adds to the complexity of developing Quantum solutions.

Grover's Search

The archetypal quantum search algorithm is Grover's algorithm [40], a search method that exploits quantum properties in a way that potentially could result in quadratic speedups.

Search problems can be split into two general categories based on the amount of knowledge about the structure of the solution space. A problem is structured if any amount of information about the solution space is exploitable in almost any way. If nothing is known or no assumptions can be made then the problem is unstructured.

We can consider the example of looking up a name in a phone book with a number (N) of names. With an alphabetized phone book, after looking at a single value we can work out the direction where our target name is in relation to it. We can then rule out all the values in the other direction. This allows our search to remove multiple values from the search pool with a single check. We can exploit this with search algorithms to minimize the amount of time. In a randomly indexed phone book, looking at a random value only tells us if it is our target name. We can't use that information to rule out any other values. In fact, this method of random guessing is the best possible method of classical search of unstructured problems with an expected time of N/2. While this seems initially feasible, many problem's search spaces do not scale linearly with the size of the problem. A more efficiently scaling search method would be able to efficiently solve problems whose search space scale more poorly.

On a quantum computer an unstructured search problem can be solved in $O(\sqrt{N}$ using Grover's search method. This is the fastest possible a quantum computer can solve an unstructured search. This is a quadratic speedup and less of an improvement than the exponential speedup that a lot of quantum algorithms have over their classical equivalents. However, search spaces can easily end up being very large, which means even a quadratic speedup can greatly improve performance.

We start by defining our search space as representing an unsorted database with N entries. These must be mapped into an N-dimensional state space H. H is representable with LogN qubits. Each entry in the database can be labelled with a unique number (e.g. 0,1,.., N-1). Generate an observable, O, that acts on H with N distinct eigenvalues with known values. Each of the eigenstates of O encodes one of the entries in the database in a specific manner. The eigenstates are denoted as:

$$|0\rangle, |1\rangle, ..., |N-1\rangle \tag{2.1}$$

With respective eigenvalues of

$$|\lambda_0\rangle, |\lambda_1\rangle, ..., |\lambda_{N-1}\rangle \tag{2.2}$$

We then take a unitary operator, U_ω, to act as a check routine that takes in inputs from the database and returns whether they meet the criteria. While the specific nature of the check routine doesn't matter, it must be a quantum function with the following two properties:

$$U_\omega |\omega\rangle = -|\omega\rangle \tag{2.3}$$

$$U_X |\omega\rangle = |X\rangle \text{ for all X} \neq \omega \tag{2.4}$$

2.3.2 Quantum Artificial Intelligence

Quantum neurons have been developed and successfully connected to neural networks. [85] A Quantum neural network can be trained in the same way a classical one can be. The structure benefits from treating it as a black box with a classical control structure providing rewards based on the outputs. This acts as a reinforcement learning structure. Quantum approaches have two advantages over classical ones. Firstly, a Quantum network can express its future states as a superposition. This allows it to probe them simultaneously, lowering the complexity of one of the main sources of computational cost. Secondly, the parallel nature of Qubits allows multiple neurons to be updated at the same time, with a much smaller growth in computational cost as the number of neurons increase. Overall, initial benchmarks have shown that Quantum neural networks can perform better than classical ones in certain cases.

A recent review of quantum artificial intelligence [32] has been highlighted in as a promising topic to look into.

2.3.3 Quantum Simulation

Motivation

Quantum simulation [20] allows the study of quantum systems that are difficult to study with real particles in laboratories and also difficult to model with classical computations. The study of these quantum systems can map molecular behaviour allowing work on chemicals required for chemical manufacturing, medicinal development or predictive physics engines. They are particularly useful at solving many-body systems, which are otherwise very difficult.

Computing a state

A quantum system's state at a given time (t) is represented as a combination of a state vector ($|\psi(t)\rangle$) and it's dynamics. The dynamics are determined by assigning a Hamiltonian (H) which is a Hermitian matrix. This system then evolves over time using the Schrodinger equation:

$$i\frac{\partial}{\partial t} |\psi(t)\rangle = H |\psi(t)\rangle \tag{2.5}$$

For time independent systems, this equation can be solved with the following:

$$|\psi(t)\rangle = \exp(-iHt) |\psi(t=0)\rangle \tag{2.6}$$

This means that using the system at when t = 0 ($\psi(t = 0)$) we can calculate the new state of the system at a specific time t by multiplying it by e^{-iHt}. However, this requires the Matrix H to be exponentiated, which we can do with a Taylor expansion. However, this is the stage that makes classical computing costly. There is no efficient way to classically exponentiate large matrices. We, therefore, rewrite the above equation focusing on the matrix by consolidating $-i$ and t into a single coefficient. This results in the calculation below:

$$U(\lambda) = \exp(\lambda\, H) \tag{2.7}$$

The operator $U(\lambda)$ acts upon the entire system, which makes it difficult to implement as a single hardware operation. Fortunately, many Hamiltonians of interest can be written as the sum of L local matrices H_j, describing interactions and state transitions of smaller parts of the system:

We now have the operator $U(\lambda)$, which acts on the entirety of the system at once. It is useful to break it down into calculations that can be more easily implemented as single hardware operations. When possible, the easiest way to do this is to break down the Hamiltonian into the sum of a number (L) of local matrices (H_j) which describe interactions and transitions of parts of the system. This is possible on many interesting Hamiltonians and is one of the key methods of quantum simulation. This provides us with the following equation:

$$H = \sum_{j=1}^{L} H_j \tag{2.8}$$

This results in our calculation of:

$$U(\lambda) = \exp(\lambda\, H) = \exp\left(\lambda \sum_{j=1}^{L} H_j\right) \tag{2.9}$$

Implementing this on a quantum computer means decomposing the exponential of the matrix sum into a set of quantum gates that can be efficiently represented by quantum gates to within a certain degree of error. This is complicated by the fact that H_j are matrices which do not necessarily commute (i.e $\exp(A + B) \neq \exp(A)\exp(B)$).

Problem Specification

Quantum simulation, therefore, leaves us with a clear problem for circuit synthesis. The aim is to generate a sequence of quantum gates ($\{P_k\}_{k=1}^{m}$) that approximate the equation $U(\lambda) = \exp(\lambda\, H)$ to within a degree of accuracy (ϵ).

$$\|\exp(\lambda\, H) - \times_{k=1}^{m} P_k\| < \epsilon \tag{2.10}$$

The matrix norm $\|\cdot\|$ returns the largest singular value; it describes the deviation between the equation $U(\lambda) = \exp(\lambda H)$ and an approximation of it. This metric provides an appropriate indicator of the error between our target matrix and our approximation; it will be used in this paper as a cost function.

2.3.4 Making Quantum Algorithms

Algorithms are as core to the development of quantum computers as they are to classical ones. However, designing them is often much more complicated. In order to exploit the advantages of a quantum computer, they must take great care to work alongside the quantum mechanical properties. Unfortunately, these are not easy to understand at the best of times. In addition, the algorithm design is greatly limited by what is currently plausible. There are several great theoretical quantum algorithms but very few have been implemented successfully. Here, we present an overview of what a quantum algorithm looks like. This will be revisited later with a look at specific tools and languages used to develop them.

A Quantum Algorithm consists of two parts, it's Qubit's states and a set of transformation gates. This ends up looking very similar to a classical algorithm. The main difference is that the Qubits exist in parallel with one another. In order to describe a Quantum algorithm, what needs to be described is the initial state for each Qubit, a set of transformations for each Qubit and a description of when each of those transformations is applied. For this reason, these algorithms are generally described diagrammatically. These diagrams map to line-by-line code implementations, with a clearer representation of what the program is doing.

As an example, a simple algorithm is shown below. This first initializes two $\mid 0\rangle$ state qubits. One is then negated while (the green X) at the same time one is turned into a superposition (the blue H). After this, a conditional two-qubit gate is applied to both. This is a conditional not-gate (the blue + attached to two lines), which flips the bottom qubit if the top Qubit is currently a 1. This creates an even chance of a 10 or 01 output, but no other output. This output is then measured and output as its classical equivalent. The measurement is marked by the pink squares. This can be seen represented in QASM as: apply a superposition (h) to Qubit 1 (q[0]); apply a not (x) to Qubit 2 (q[1]); apply a controlled not (cx) to Qubit 2 based on Qubit 1. This is then measured and stored in designated classical registers (c[0] and c[1]).

2.3.5 Shor's Algorithm

Shor's Algorithm [76] is a method of factorizing integers in polynomial time, which makes solving large integer factorisation possible. The best classical methods can only do this in superpolynomial time, making them take too long to be considered a feasibly applicable solution. This is important as major components of cryptography rely on

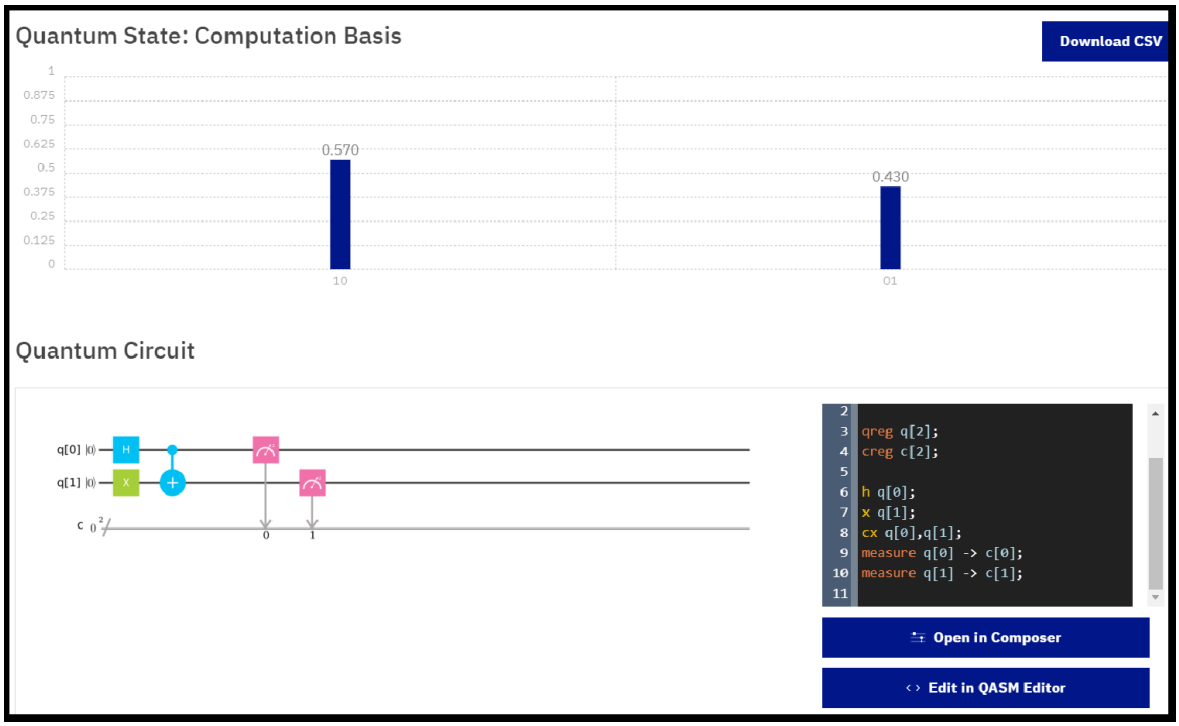


Figure 2.4: *An Example of A Simple Quantum Algorithm Written with IBM-QE's Online Interface*

very large integers not being factorizable. This includes the most widely used cryptosystem, RSA. This highlights one of the most exciting parts of quantum computing, the idea of breaking classical barriers. RSA's ubiquity in security systems demonstrates a huge amount of trust in it, trust that would not have been able to be brought into question without quantum methods.

2.4 Quantum Computing as an Example of Mixed Hardware Computing

Quantum computers exist as one of the most well developed non-classical models of computation. As shown above, quantum machines have a large set of methods of representation that allow programs to be accurately and precisely defined. In addition, there is an interest in quantum computers on an industry level. This takes the form of several standard development and compilation methodologies, providing a meaningful framework to test and demonstrate any improvements aimed at Quantum machines. As a final note, a number of quantum machines exist as mixed-hardware machines. That is to say, quantum machines in large-scale public use have different components of their hardware, in this case, different qubits, that have sufficiently different properties that considering these differences are likely to cause a significant increase in quality. In

total, this makes quantum computers a natural and useful example of mixed-hardware computation.

2.4.1 Other Kinds of Mixed Hardware

In most standard models of computing, there is an assumption that the hardware is uniform across the machine. In these models no matter which piece of the hardware is used, the result will be similar enough to not be worth considering. This is an effective model for almost all classical, transistor-based computations. However, when considering non-classical systems (such as quantum machines) or performance-sensitive classical machines (such as large GPUs) the differences can be enough to provide significant speedup if taken into account. While any system will have some hardware differences, we define mixed hardware systems as only those where there are significant differences between the quality of parts of the hardware.

A hybrid computer is a system made from two or more different types of computing paradigm that functions as a single whole. A subset of these exist, called heterotic computers [48], that definitively specify that the hybrid computer's capabilities must surpass the sum of its parts. While they're only recently being formalized, hybrid computers have been ubiquitous in various forms for a long time. As we approach the limits of what standard transistors can accomplish, it's becoming more important to look at alternative methods to improve the computing power we have access to.

Hardware with a single substrate can still rely on mixed hardware. The simplest example of this is a CPU attached to a GPU. The CPU will perform better on most small-scale single tasks but will be outperformed by the GPU on larger repetitive tasks. In quantum computers, mixed hardware is a natural byproduct of the architectures. Qubits must be connected to in order to perform certain operations. However, these connections are fragile and complicated so it is not possible to connect every qubit with every other qubit. In addition, due to the sensitivity of qubits to error, differences in the physical location of qubits and connections can have a noticeable impact on their fidelity. Similar to classical machines, some tasks will perform better (or only be possible) when run on some parts of a quantum machine.

Why Aren't Other Mixed Hardware Models Used?

Mixed hardware computers suffer from the same lack of formalization that most unconventional computers tend to have [47]. Most of the classical mathematical and abstract models for formalization were created alongside classical computers and rely heavily on the assumptions of their physical representation. This makes it a non-trivial task to apply almost any of these formal methods to a Mixed hardware computer in such a way to properly exploit their computational power. While they can be done in a rough way, these don't preserve the fundamental properties of a classical implementation.

Most notably, this rarely preserves the efficiency of the classical equivalent making it difficult to gain an increase in performance from these methods. Of particular note is the difficulty in predicting fidelity (i.e. low error properties) in mixed hardware devices when using models that assume uniform hardware.

The added complexity of mixed hardware models makes implementing them in a useful way difficult. Mixed hardware models are already less well developed than standard models. Classical computation is based on Turing machines and a binary representation of data. This has worked well for conventional computers so far, as their physical components lend well to this type of encoding. However, unconventional models do not necessarily lend well to a binary system. This can lead to two problems. The first is that finding a replacement encoding method is often very difficult. This can cause difficulties in developing algorithms where standard principles either do not work in the encoding or must be rebuilt from nothing.

Assuming an appropriate encoding method, there can still be challenges with the lack of familiarity with the system. Representations are used to understand and explain how a program is working and what it needs to run fully. Complicated representations [57] add in an accessibility barrier that can greatly slow down the development of algorithms, languages and programming tools. An example of this exists in quantum computing. These are often represented as matrix operations which on a large scale cease to be human-readable. Graphical representations of these operations are often used in software tools as they have the same representational power but are much simpler for humans to comprehend.

This can commonly be seen when unconventional methods are adapted to attempt to simulate a binary code. Attempting to use familiar conventional methods on unconventional encoding often results in a final product that cannot fully utilize its computational advantages. On the other hand, attempting to develop algorithms in radically difficult paradigms can be quite difficult. Both of these can be clearly seen in the development of quantum algorithms, which has had a vast quantity of methods attempted to various degrees of success.

2.5 IBM's Quantum Experience in Depth

IBM's Quantum Experience (IBM-QE) is a set of several simple Quantum computers made publicly available by IBM. [4] These are 2 five-qubit computers, 2 sixteen-qubit computers, and a single twenty-qubit computer. These are made available with a cloud control structure. This means that quantum algorithms can be run remotely. An interface is provided to allow the submission of QISKit code. In addition to being able to run it on physical quantum computers, it can be simulated or displayed as a visual circuit. There are currently no costs associated with using these devices, however, use

is limited if not officially working with a research group or not publishing papers. The programming language used, QISKit, sufficiently allows the development of the hybrid code. It provides functions for creating both quantum and classical registers and allows functions from both to be alongside each other. Data about the current runtime of the program is available through a function, allowing the number of registers, run-time and step count to be looked at. This provides sufficient metrics to be able to compare the performance of different algorithms.

Currently, full details of the hardware and software used are publicly available [5], as well as a provision of resources to help support user verification. Furthermore, complete tests can be run on both the physical computer and a simulator. This generates a clear comparison of both the theoretical and actual computations. Practical sets of tests can be implemented to empirically show any theoretical improvements. Currently, there exist several standard algorithms, including a set of simple benchmarking tests for quantum computers. While the IBM-QE is too small to be considered a NISQ, it is built on the same principles. This means any developments that can be tested on the IBM-QE are, by design, able to be scaled up to work with larger NISQ computers.

2.5.1 openQASM

Open Quantum Assembly Language (openQASM) [26] represents an intermediate level specification of a quantum program The typical journey of a quantum algorithm goes through several stages of development. Quantum programs require a mixture of quantum and classical computation and a great deal of complicated coordination. Some of this is a mix of purely theoretical design while others refer to concise machine instructions. At all levels, the idea of an intermediate representation is useful. An intermediate representation is a language or model between the layers of a source language description and hardware-specific machine instructions. QASM exists to fill this niche in quantum computers.

openQASM's language describes universal physical circuits. It does this through a collection of arbitrary single-qubit gates and a single two-qubit entangling gate (CNOT). Larger gates can be described by specifying them in terms of built-in gates and other previously described larger gates. In addition, several components of the quantum-classical interface are also programmable. Specifically, measurement, classical feedback and state reset. Higher-level programming primitives are not present at all. The only available types are classical and quantum registers. These are one-dimensional arrays of bits and qubits. When initialized, qubits are set to state 0 and bits are initialized to 0. The default gate basis is "CNOT + U(2)". This means there is only a single available two-qubit gate, the CNOT.

Each single-qubit unitary gate can be described with the following function:

$$U(\theta, \phi, \lambda := R_z(\phi)R_y(\theta)R_z(\lambda) = \begin{pmatrix} e^{-i(\phi+\lambda)/2}cos(\theta/2) & -e^{-i(\phi-\lambda)/2}sin(\theta/2) \\ e^{i(\phi-\lambda)/2}sin(\theta/2) & e^{i(\phi+\lambda)/2}cos(\theta/2) \end{pmatrix} \quad (2.11)$$

Where $R_y(\theta) = exp(-i\theta Y/2)$ and $R_z(\phi) = exp(-i\theta Z/2)$.

Outside of gate functionality, measurement is also possible. This measures the qubit in the Z-basis and stores the corresponding value to an inputted classical bit. Measurement corresponds to a projection of one of the eigenstates of Z. In addition, qubits can be reset to $|0\rangle$. This represents a partial trace of those qubits before resetting them. There is also a barrier to instruction. This prevents gates from reordering across their source line. Finally, there is a single classically controlled quantum operation. This is a simple if statement. This runs a quantum operation based on a value stored in a classical register. This is useful in allowing the measured values of quantum operations to dictate the future behaviour of the circuit.

2.5.2 QISKit

The IBM-QE's software core is based on a language called QASM, which has been further developed into a developer framework called QISKit [25]. QISKit is the current standard for development for code run on IBM-QE. QISKit is currently divided into two separate libraries, Terra and Aqua. Terra provides the foundations of low-level development. It provides tools for developing quantum systems in terms of physical circuits and pulses. It's built for optimizing around a specific hardware's constraints. Aqua is a higher-level language. Aqua is meant for application development, particularly for people who are not experts in quantum hardware. It can both assume defaults for hardware or take a Terra input for configuration. Aqua provides methods for translating certain classical algorithms or code fragments into quantum code. Overall, this provides a layered structure to create a design flow. This was previously done with several different languages, instead of a single one.

In addition to these currently available libraries, two more are in development, Ignis, and Aer. Ignis is aimed at reducing noise and error correction. Ignis is partially implemented as a part of Terra. Aer is meant for accelerating development by introducing surrounding tools, like debuggers and simulators. Most notably, this aims to include verification tools that are currently absent in most areas.

2.5.3 Why IBM-Q is the Most Appropriate Technology to Target

IBM-Q provides the necessary functionality to build our experiments, while its availability pushes it as the most appropriate technology. The hardware is publicly available

for running tests on a cloud-based server. As part of our focuses is making methods more approachable for non-experts, it is, therefore, useful to have hardware that is also available to non-experts. In addition, IBM-Q provides comprehensive benchmarking and hardware detail. As one of our focuses is real hardware, being able to look at the full details of a commercial machine ensures that our benchmarks are built in reality. Finally, IBM-Q does have a software stack to work with. While there are still holes, it makes more sense to build on the current methods rather than rebuilding from the ground up.

Chapter 3

Literature Survey: Program and Circuit Synthesis

3.1 Introduction to Program Synthesis

Program synthesis is an automated process of generating code that aims to assist non-experts in the implementation of algorithms. A program synthesis problem provides a program specification and is solved by a generated set of code that implements this specification. This enables a user to implement code based on their ability to describe the expected behaviour and output without necessarily understanding the underlying structure. It can also be used in complicated and difficult domains where users may be able to provide certain expertise but may not understand the full system. Even in cases where full expertise is expected, automated generation of code still provides an increase in ease and speed of code production. Automation is especially relevant in fields with inherent difficulty or complexity where experts will be rarer and more expensive.

The standard use case for program synthesis is allowing non-programmers to build programs with minimal training. Spreadsheet and database management is an integral part of any business. Even the most common (Microsoft Excel) have certain features locked behind barriers that must be overcome with domain expertise. Users are much more comfortable using automated tools instead of pure programming measures. An example of this is Microsoft Excel's Flash Fill feature, which manages the syntax and structure for a user that knows the intent of the functions they want to use.

It is also worth noting that the types and duties of people who are non-experts can be vital to a business's function. Lack of programming expertise can exist throughout a business hierarchy, including high-level managers, administration teams, and analysts. A job not requiring programming expertise does not mean it is a lesser or unimportant task. Providing users in these positions with simpler tools allows them to perform a

wider range of tasks faster than they would have otherwise been able to.

In complicated systems, it is common for teams to have members who are experts in parts of the system but not others. In these areas, we can expect high-quality descriptions of behaviour and functionality but the users may not be aware of the subtleties of the system. Program synthesis provides a method for partial experts to describe only their area of expertise, with the implementation then provided automatically. This can save teams with already complicated sets of skills requiring even further expertise, simplifying the implementation process. An example of this is control-system engineers building device controllers. These require in-depth knowledge of the interactions between the real system and the discrete controller. Automated program synthesis helps control-systems experts make these controllers despite not being experts in the car, toaster or other highly specific device being designed.

Program synthesis is still useful for software experts as the automation can be used to speed up or avoid tasks that are time-consuming. Practical software development comes with a pressure of time and cost, often leading to corner-cutting. Technical debt [51] refers to the cost of having code that is bloated, complex or otherwise inefficient even if perfectly functional. Refactoring this code is expensive and difficult for humans to do. However, functional and professionally developed code acts as a specification for itself. Program synthesis here can be used to re-implement this code with a focus on non-functional properties, This allows a kind of program synthesis that is not a problem to be solved but instead optimised.

3.1.1 Quantum Circuit Synthesis

Developing Quantum Circuits is an opportunity for synthesis technology that would allow experts with software backgrounds to have a less complete understanding of Quantum physics. Quantum Computing development requires expertise from a wide breadth of fields including a number of different, complicated and often highly technical areas of physics, software engineering and theoretical computer science. Easing the required expertise of each individual helps to expand the pool of people able to meaningfully contribute to the field. We can also see that this situation overlaps with one of the areas of research done on classical machines, under the “partial experts” umbrella. We can draw parallels between control systems engineers working on arbitrary physical systems, where we can expert certain domain knowledge but cannot expect full knowledge of the technical minutia from all developers. This indicates that quantum computing is likely to benefit from adapting these classical techniques.

3.1.2 Automated Circuit Synthesis

Lee Spector's Book

Lee Spector's book is an influential early motivator for work into the automatic quantum program. While not the first work, it provides a comprehensive summary of promising methods in the field. What sets it apart from previous work is its focus on tangible achievable methods with examples of useful outcomes. This changed the perception of automatic quantum programming from an abstract but promising field to one with a clear, tangible roadmap. Here we summarize this roadmap and expand on how this relates to the modern state of research.

Genetic Algorithms

The phrase "genetic and evolutionary computation" is commonly introduced with the definition of any algorithm that borrows from genetic or evolutionary principles. This is, of course, an intentionally loosely worded definition that fits in a wide array of methods and programs. While we don't aim to provide a fully encapsulating definition that covers every algorithm ever designed, it is worth examining an overview of the structure and highlighting core components that show up in the examples relevant to this thesis.

We define genetic algorithms (GA) as any algorithm with a structure that contains the following steps. First, a number of solutions called individuals are randomly generated to form a set called the population. Then, each individual from the population is assessed using a function that measures its fitness or value. The fitness function may be exact (e.g. length of solution) or a heuristic (e.g. neural network assessment). Using this assessment, a selection of the best solutions are taken. This selection is then modified by a set of mutation functions with the purpose of increasing variation within the population. The cycle of assessment, selection and mutation is repeated a number of times, afterwards which the best solution is chosen.

One of the key advantages of this is the scalability of GAs over other search methods. The main computational cost of a GA lies in the calculation of the fitness function. However, each fitness function is independent of each other fitness function. This independence means that each fitness function can be calculated in parallel. This is especially relevant to quantum programs. The assessment of a quantum program is likely to involve some measure of quantum simulation, which means the use of a quantum machine. Quantum machines are inherently parallel and therefore can more easily work with algorithms that can be run in parallel.

Genetic Improvement

Genetic Improvement (GI) is a category of GA where each individual is an entire executable program. This tends to include a few features unique to them which aim to

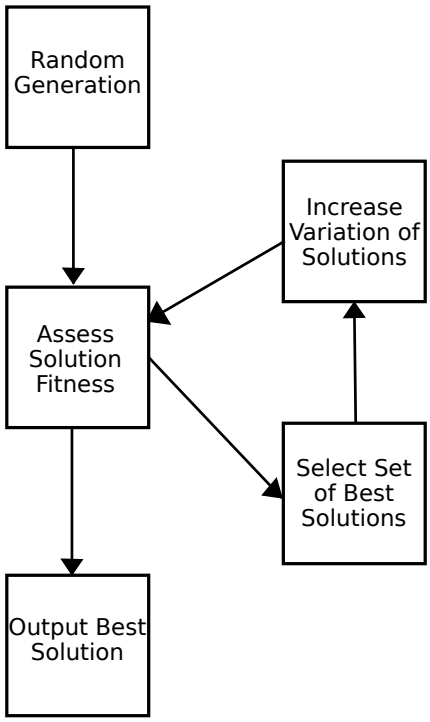


Figure 3.1: *A model of the standard GA algorithm structure*

represent the structure of a program. The individuals of a GI incorporate hierarchical and composite structures that exist in programs, as well as the requisite flexibility in length required to implement them. In addition, including functions means including both general case operators but also more specific problem-specific operators.

An advantage of GAs, in general, is that they require less knowledge about the structure of their problem than other search methods. This is highly relevant in GI methods being applied to quantum programs as we do not have a strong understanding of how to build quantum programs. For a GI method, we can define a quantum program by the expected functionality. This is a much more realistically achievable short-term goal that would allow the efficient building of quantum algorithms. This can be made especially easier if the initial population is not randomly generated but instead built from existing quantum programs. This would let us use existing human insight then extend on it with clever automated search.

This highlights the key areas and objectives of automating quantum programming. The aim is to develop methods that can build or improve quantum programs without a full understanding of the system. They take advantage of the fact that we can provide high-quality descriptions of functionality but are also not as limited by our lack of complete understanding of quantum program design. In addition, they provide advantages to efficiency when scaled up to larger problems. These features are useful

to keep in mind when looking at other methods of automated program synthesis, as the goals and limitations of the field will still apply.

3.2 Software Stack Engineering

3.2.1 Introduction to Software Stack Engineering

A software stack is a collection of independent components that work together to support the execution of an application. To execute with a software stack, the execution is moved through layers one at a time. In principle, each layer should be independent of the previous. The components, which may include an operating system, architectural layers, protocols, runtime environments, databases and function calls, are stacked one on top of each other in a hierarchy.

It is necessary to translate high-level algorithm descriptions into a set of operations that can be physically run by a quantum machine. This is known as quantum circuit compilation (QCC). Compiling a quantum circuit requires reorganizing or adding gates to ensure qubit states are in locations where all operations could be performed with the physical constraints of a specific quantum processor. This process is often done in several steps, moving through layers such as completely abstract definitions, human-readable code and machine instructions before being translated into physical operations. A software stack is simply a map of these layers as well as the techniques and tools used to move across these layers.

The software stack provides a variety of information about how the final circuit is formed. Moving through each layer requires the translation of the previous layer to a new representation. This introduces errors that can be calculated, characterized and mitigated. In addition, the different layers of a software stack describe relationships between the components in different ways. This means that some connections can only be seen on specific layers of the stack. This is important as errors conflate with each other, meaning that knowing how errors interact or correlate can allow us to perform more effective mitigation measures.

The purpose of a software stack is to break down the work of a large piece of software into smaller more manageable chunks. This provides an increased amount of flexibility when progressing through the structure. Problems or changes made to layers within the stack can be contained within them, meaning it is not necessary to understand exactly what every other layer is doing. We highlight one of the advantages of this for quantum circuits being that field-specific expertise required in one layer can be kept away from other layers.

We use QISKit as an example of our quantum software stack. The top-level of

QISKit is the input generation, performed manually. This is translated into an equivalent quantum solvable problem, such as a Hamiltonian or a Grover oracle. This is then coded as a QISKit Aqua algorithm description. This provides an algorithm level description of the code. This is then translated into QASM to generate a circuit mapping. This circuit mapping is then translated directly into hardware operations.

3.2.2 Software Pipelines

While the classical view of a software stack looks at the total development cycle, it is also worth looking at smaller stacks within that. These share many of the same structures and properties but are focused on smaller steps of the process. These smaller structures are often configurable and provide tools for optimization within the process. These are referred to as software pipelines.

3.2.3 Quantum Software Stack

Quantum algorithms work by applying quantum logical operations (called quantum gates) to a set of qubits. These are analogous to the logical gates used in a classical program. A quantum algorithm represented solely in terms of quantum gates is called a quantum circuit. These algorithms are currently specified on idealized hardware, which ignores the details that are specific to actual types of hardware. While this is helpful to programmers who are building the program, it does mean that they cannot be directly run on any given quantum computer.

It is therefore necessary to translate a high-level algorithm description into a set of operations that can be physically run by a quantum machine. This is known as quantum circuit compilation (QCC). Compiling a quantum circuit requires reorganizing or adding gates to ensure qubit states are in locations where all operations could be performed with the physical constraints of a specific quantum processor. This process is often done in several steps, moving through layers such as completely abstract definitions, human-readable code and machine instructions before being translated into physical operations. A software stack is simply a map of these layers as well as the techniques and tools used to move across these layers. We highlight one of the advantages of this for quantum circuits being that field-specific expertise required in one layer can be kept away from other layers.

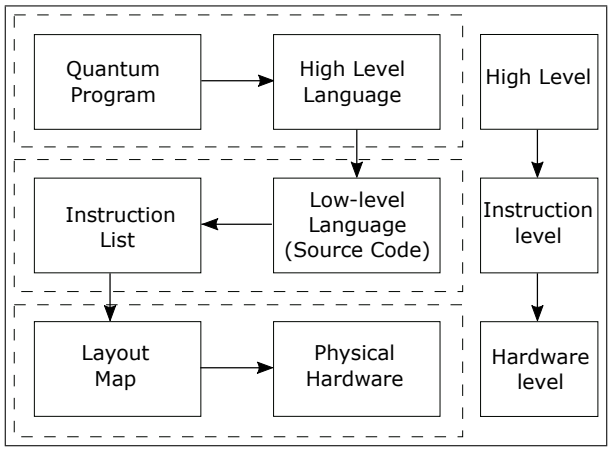


Figure 3.2: *The Quantum Compilation Software Stack. The left hand side shows the pathway through stages while the right hand side shows the abstraction level of the stages.*

We look at a description of an idealized quantum toolchain [35]. The quantum stack starts with a high-level description of the algorithm. The high-level description is typically the human inputted section of the stack. This description is then broken down into its source code to generate a list of instructions to be completed. The instructions are then mapped to a low-level physical layout map which can be run on a physical circuit. Overall, the algorithm exists in five states across their compilation: high-level language, low-level set of instructions, circuit-map, layout map and physical hardware.

3.3 Refinement and the Software Stack

3.3.1 Formal Development By Refinement

The basis of formal verification by refinement is the creation of an abstraction that reflects the properties of a software system. [95] Provided the abstraction accurately reflects the system, it allows proofs to be generated from the abstraction that can guarantee or prevent the behaviour in the implemented system. However, it can be difficult to prove that an abstraction truly matches a given software system. Refinement based verification is one approach that aims to link abstractions to implementations. Refinement based development is the idea of starting with a theoretical model and adding features in iterative steps until a final implementation is reached. Each iteration of a formal refinement approach is motivated by a formal relation between the stages that precede and follow it. This relation is normally one that tries to encapsulate a "correctness of equality"; This means that each iteration should do the same things in the same way as the connected steps. Each step provides a new instance that continues the chain, creating a further relation from the top-level abstraction to the fully

implemented system. This ensures that eventually, the final coded implementation will have a full chain of proofs back to the original abstraction. This property provides a great deal of utility when relating proofs from the abstract states to that of the final implementation.

One of the original research goals for this project was to investigate refinement approaches to improving property preservation. Refinement approaches are strict verification methods. They guarantee properties by translating strictly equal properties through strictly correct translation rules. However, it is clear that in order to make a practical improvement in NISQ error devices, compromise is often key. It is this principle of compromise that raises the question of whether exactness is a property that is required or whether fuzzier methods are more appropriate. We, therefore, look at the advantages of exact refinement as well as the costs and restrictions of refinement. These can then be taken into the context of developing a quantum software stack that takes the best advantage of refinement principles.

Refinement verification's core purpose is to ensure that a high-level model is correctly matched by a low-level implementation. Refinement allows a large verification task to be broken into smaller, more approachable, chunks. Model-checking can be applied to each of these chunks separately, with the confidence that the sum of their parts will still be accurate.

When it is necessary, refinement provides one of the easier methods of verifying systems. However, when not necessary it can be difficult to justify. Refinement is expensive, from the perspective of both time and monetary cost. This is primarily because refinement is difficult and requires a particular skill set and background. While this isn't as large a problem in academic areas when introducing a methodology into industry cost and ease of use are huge focuses. Quantum computing is already a difficult area to understand so putting multiple expertise barriers greatly limits algorithm development. This difficulty also raises issues in flexibility. Methods included in refinement systems must have translation paths to each other with provable equivalencies in order to be useful. This places a barrier before using any method for which those do not currently exist. This may simply be the cost of demonstrating the proofs or it might be that this proof is not possible at all.

Overall it is difficult to justify a pure refinement approach. The difficulties added in development all but ensure that any pure refinement approach would struggle to make an impact in mainstream quantum code development. However, it is still useful to look at the advantages of refinement and ensure that these are preserved in the approach used. The main one of these is that refinement allows larger systems to be broken down and dealt with in smaller chunks with the certainty that the larger system will be correct if all chunks are correct. This iterative structure is still useful in quantum code development, without the exactness constraint. We can consider the

implementation of a quantum algorithm as an iterative process of translation steps from abstraction level to physical implementation and generate the same structural advantage of refinement. However, the argument that these translation steps should be exact does not hold. Instead, translation steps that are correct to a high-certainty or within certain bounds are a more appropriate measure. We refer to this order of languages and translation steps as a software stack with the main distinction being that the output is not the result of a verification methodology but a software engineering one.

3.4 An Analysis of the Gaps in the Research

A More Practical View of Quantum

Quantum computing research is focused on theory and proof of concept over practical implementations of algorithms. We see in the literature [82],[82],[82] that it is common for quantum research to explain examples on idealised quantum circuits and not provide code implementations of their methods. For recently published work this is expected - quantum machines are historically only a few qubits large, too noisy to expect a result and lacking tools to usefully develop on.

It is also worth considering the github topic pages for quantum computing compared to other computer science fields (for example genetic algorithms, chosen for familiarity). It can be seen that the commonly accessed and used quantum computing software repositories are frameworks for development or tutorials for basics, with very few implementations. On the other hand, genetic algorithm has a much higher rate of attention to pages which demonstrate practical implementations of GA.

The latest quantum machines [5] have more qubits with error rates that allow simple implementations. This allows a more practical and empirical approaches are possible. Example implementations, even if only built as toy solutions can be reasonably built and run as is expected in wider computer science research.

Quantum Computers are Not Theoretical Objects

It is currently difficult to incorporate real quantum machine information into quantum approaches. Not only are there multiple kinds of quantum hardware (circuit-based, annealling, hybrid) that are radically different from each other, even within those models there is large difference. Publically available circuit-based models have thoroughly measured values for these properties, including average error of individual gates and qubits. However, the sheer complexity of this information means that it rarely appears in modern research papers. Instead, it is often treated with basic but useful assumptions (for example “lower average error is better”). These assumptions save sufficient time over hand choosing but do provide inaccuracies. Developing methods of automatically considering this can remove these inaccuracies without requiring users to spend time hand picking algorithms.

Quantum Software Engineering is Underdeveloped

Classical software engineering is far ahead of quantum software engineering. Classical models of testing, language design and function abstraction has been developed for decades. Quantum computing theory is a much newer subject and simply does not have this ground work. However, with modern quantum languages being developed (such as qiskit) it is important to look at classical software engineering principles before attempting to develop new methods. In particular, we look at the flexible, modular approach that classical function design uses. Trying to emulate this with quantum circuit design is important to create software that can be applied to a variety of machines and problems.

Quantum Development is Still Too Hard

Quantum software development is currently very hard, relying on a wide range of knowledge of very specialist subjects. Mainline quantum software like qasm needs to hand-allocate gates to qubits to build an algorithm for a specific machine. Even assuming a background in both the quantum particles involved in the hardware and the algorithm's computer science theory, there are a great many details of the error rates, qubit arrangement and implementable gate sets. These properties change how a user would build an algorithm in ways that are non-trivial and non-obvious. There is a gap in approaches designed for simplification of the process, not just performance!

3.5 Promising Methodologies to Address the Gaps

3.5.1 Modelling Practical Quantum Computers

Including Hardware Details in Quantum Models

NISQ-era machines utilize inherently noisy hardware and should be modelled as such. Not only is each quantum computer different, different qubits have different noise parameters. These error differences are benchmarked and often available, however, they are not used when building quantum programs. Providing a model that integrates these error profiles is the first step to building a more practical foundation of quantum software development. This foundation aims to lay out more clearly the properties of real physical computers, not theoretical models.

Running Real Tests on Real Machines

In order to check whether a model of a quantum computer is accurate, it is necessary to compare it to the real machine. For this reason, an emphasis has been put on ensuring that the approaches being proposed can be run on real technology. This provides an insight into how accurate the methods are. If the model predictions differ greatly from real performance, then they are not useful. In addition, they present a measure

of feasibility. Any implemented system acts as proof that it can be built and these provide useful examples to show that these models work.

3.5.2 Automating Quantum Programming

Genetic Improvement

One area within the field of Genetic Algorithms is Genetic Improvement. GI can be used as a method fir program synthesis by taking existing programs and uses automated search to generate better programs. The automated search typically works by applying small changes to the program and selecting the results with the best fitness, in the same way GAs work. GI has been used previously to modify software to better run on different hardware [52]. It is therefore the obvious candidate for any new methodology modifying quantum programs across hardware.

Temporal Planning

Automated planning is a branch of Artificial Intelligence that aims to solve specific objectives by selecting an ordered series of actions to perform. [56] These problems are defined by a description of the initial state, a set of end goals and the set of actions that can be performed. The simplest case of these is called the Classical Planning Problem. These problems require a single known initial state, actions that are instantaneous and deterministic, and only a single track to perform actions.

Temporal Planning Problems are an extension of Classical Problems that allow the expression of more complicated problems. [9] Specifically, Temporal Planning introduces the idea of a meaningful time state by providing actions with a duration and specific time to occur. It is also possible to directly specify what resources are used when over the duration of an action. With a better model of resource availability, it is possible to allow multiple actions simultaneously, provided the resources are available at the time.

PDDL (Planning Domain Definition Language) is a formalism that aims to standardize AI planning problem descriptions. [36] These models can then be given to planner programs that attempt to solve them. PDDL defines models in a two-part fashion. The first describes the larger context of the problem, called the domain. Domain definitions, at minimum, consist of predicates and operators. They may also commonly contain definitions of typing, constants, and functions. The other part is the problem definition. This describes the specifics of a particular value. This includes exact values for the initial state, objects present and the goal state. Multiple problem definitions can be written for the same domain.

PDDL is overwhelmingly the most popular planning domain language. Popularity is important when working with planning models as the AI planners themselves must

be able to parse the problem. A more popular planning domain means a wider range of planners can be looked at. Importantly, the current leading planning work comes from the International Planning Competition which uses PDDL. This makes PDDL the obvious choice of language to access the largest range and state of the art planners.

3.5.3 Current State of the Art

The current state-of-the-art in quantum circuit compilation with temporal planning looks at generating circuits with the smallest possible time from the start to the end of the computation (called the makespan). This equates to minimizing the largest distance between the start and end of a pair of operations. In addition, the makespan was measured as the longest makespan of a single qubit rather than the whole circuit. This means that the metric being minimized is the largest distance between the start and end of a pair of operations where both operations are applied to the same circuit. Based on these methods, an initial set of problems were gathered and circuits were compiled with minimum makespans [89].

They define the problem with an idealized quantum circuit and a hardware architecture specification. An idealized quantum circuit is a set of qubits and operations to be run on these qubits. In addition, there is an explicit specification of which gates commute (and therefore can be reordered). A hardware architecture specification is a diagram containing the position of qubits and which operations can be applied to them. The objective is to reimplement the idealized quantum circuit with an order that is valid on the hardware model while minimizing the makespan. Other works have improved on this problem by looking at other approaches in the wider AI field (constraint programming[12], rollout heuristics [18]).

This provides a strong theoretical foundation for the PDDL synthesis of quantum programs. However, there are still some addressable improvements. The first is that the model requires an explicit specification of which gates commute. This requires a degree of expertise in quantum systems which can be hard to attain. Targeted automation can provide an opportunity to mitigate the expertise required. Secondly, the model still looks at idealized hardware. This is useful in building the theoretical foundations for the long term development of the approach. However, in the current era of quantum hardware more practical models can be made by looking at current machinery. The short term-analysis of current machines can provide a better insight into what is needed to build the long-term foundations of quantum circuit synthesis.

3.5.4 Making Quantum Systems Easier

Learning From Control-Systems

Control-system engineering in particular has a lot of overlap with quantum engineering fields. Control systems aim to attach automated controllers to physical devices with programmed chips. This results in a discrete software controller connected to a complicated system that only provides a limited scope of interaction. The developers of these are often experts in control systems and the relevant software but are not necessarily full experts in the system which is being controlled. This is similar to how much quantum software development is structured. Methods used to clarify and automate away expertise requirements in control-systems works are therefore a promising area that might be generalized to apply to quantum development.

Automating Hardware Specifics

One of the simplest ways to lower the expertise required to work on quantum computers is to automate portions of the development. In the case of full experts, this means that they can perform these portions quicker than by hand. In the case of partial experts, it means that they would need a full understanding of the processes required to implement it by hand. Instead, they would simply need to understand the automation process. For quantum computing especially, this is a big change. For those with a software background, understanding automating tools is a common skill-set, as is having some concept of the background circuitry. However, having a more in-depth intuition about a quantum computer is much less common.

In control systems and GPU, this is most commonly seen in automating the integration of changing hardware systems. As discussed, the similarities in these fields act as an indication that quantum systems would also benefit from this. Quantum computers are each built on different hardware. These have different specifications. Automated program tuners that take in hardware specifications as inputs and use them to optimize programs exist in other mediums. This indicates that they're a promising area of focus for research.

Explore Software Engineering Techniques

The other clear difference quantum development has from other software development is in the vast gap in software engineering techniques. The exact worth of having a strong software engineering structure can be hard to precisely evaluate. Larger projects with many developers who need to communicate benefit the most, while small projects by individuals may benefit less. As quantum research increases in size and scope, these methods are becoming increasingly necessary.

We consider the main purpose of building software engineering frameworks for quantum computers to be simplifying the design process. We can narrow this down further with the methods discussed above. We aim to explore software engineering techniques to simplify the process of circuit synthesis by segmenting out the distinct areas of development and identifying any areas that are fit for automation.

Chapter 4

Thesis Research Aims

4.1 Aims

The main aim of the thesis research is to provide automated and structured methods of generating real quantum code. The aim of automation is to increase the speed and ease of development. The aim of a structured method is to allow new innovations to be slotted into a standard quantum pipeline. The focus on quantum code is because these methods are particularly relevant as the field matures and as quantum increasingly threatens to outperform classical computers. The further focus on **real** quantum code is because one of the main limiters is the relatively low quality of quantum hardware, so it's important to take these real limitations into account. The overall effect we expect is to demonstrate that practical, heuristic methods can work alongside theory to provide better solutions to real quantum problems.

Automating Quantum Code Generation

We aim to research methods of automating quantum code generation. We take methods used in classical code generation and look at how they can be applied to quantum code generation. This poses two questions: what methods are appropriate and what challenges are posed by transferring these methods to quantum code.

This is focused on in contributions 1 and 2 (chapters 5 and 6).

Structuring Quantum Development

We aim to look at applying software engineering principles to quantum development. In particular, there is a focus on ensuring that our methods work in an iterable and modular fashion. We do this by clearly laying out the total structure of circuit compilation from the circuit level. In addition, we show where our methods exist within this structure and what this position within a larger process entails.

This is focused on in contributions 2 and 3 (chapters 6 and 7).

Real Quantum Development

We aim to look at the development of real quantum systems over theoretical models. To do this, we look at ensuring our methods are tested on real quantum hardware. We also look at the differences between the theoretical models and the final circuits. In particular, we focus on a description of the steps that are taken to transform the theoretical model into a real circuit.

This is focused on in contributions 1, 2, and 3 (chapters 5, 6 and 7).

4.2 Research Project Overviews

Project 1: Automated Circuit Design

The objective of project one is to provide an automated way of generating quantum circuits to re-target hardware. We provide automation with a genetic programming technique. This technique takes a circuit and a quantum computer backend and iterates modifications to the original circuit to make it a better fit for the backend.

Project 1 looks into automating quantum code generation by providing an example of automated quantum code generation. This was a useful step for the research as it allows a practical method of probing the current methods of quantum circuit generation. Using this as an example case study, it is possible to more clearly see where the previous methods are at a disadvantage to automated methods.

Project 1 looks into structuring quantum development by focusing on localised refactoring code. Here localised refactoring refers to improving local code without changing its function in the larger codebase. One of the advantages of iterable and multi-layered software developing approaches is that single layers of development can be changed without affecting the entire method. The quantum hardware can be updated to a state of the art model, without the requirement of rebuilding the entire compilation process.

Project 1 looks into real quantum development by joining a highly theoretical method with a set of practical tests on real quantum machines. We look at the language ZX to represent our circuits. This is a language with a strong theoretic background but with only a small amount of practical research. We aim to use this to model a method to target real IBM hardware. This helps provide some insight into the difficulty of linking theoretical approaches to quantum computing with more practical ones.

Project 2: Automated Scheduling of Qubits

The objective of project two is to approach a quantum hardware management problem in a way compatible with a standard classical hardware management problem. We do

this by treating a quantum circuit generation problem as a scheduling problem. We then look at standard classical scheduling solvers. We then select one, which is automated planners. Specifically, we look at planners using PDDL models. We reformat the quantum circuit generation problem into a PDDL planning problem. These are used to run a test set of problems, the results of which are used to generate a baseline performance metric.

Project 2 looks into automating quantum code generation by looking at how classical automated code generation can accommodate quantum methods. We provide a model that is compatible with PDDL planners. This means that development in classical PDDL planners can be adapted to quantum code generation. It also further highlights the difference between and quantum and classical code generation.

Project 2 looks into structuring quantum development by looking at how the structure of developed classical methods can improve quantum methods. The classical planning problem solvers have a robust structure. New planners and new models can be built with a great amount of ease and the state-of-the-art is easily accessible.

Project 2 looks into real quantum development by considering a highly results-based area of research. Automated planners are measured primarily by planning competitions. By providing a quantum scheduling problem, we can gain access to planners measured solely by the speed and accuracy of their performance. This allows the practical realization of theoretical improvements to measurable increases in solution quality. The end result is an approach that can map real software to real machines.

Project 3: Automated Software Stack Analysis

The objective of project three is to build an automated classifier that builds a transpilation stack from a set of pre-written methods. This is done by creating an abstracted model of transpilation rules and construction rules on how methods must be chosen. After this, circuits are assessed to fit a support vector machine that predicts which methods would build the best transpilation stack. We generate a profile of the accuracy of the classifier by comparing its results to training data classified with manual execution.

Project 3 looks into automating quantum code generation by providing a method of creating initial systems to simplify the task of building transpilation stacks.

Project 3 looks into structuring quantum development by providing a method to automatically design a software stack. While this was applied to a smaller pipeline, this still provides a useful proof of concept to demonstrate the advantages of automated software engineering.

Project 3 looks into real quantum development by building a transpilation stack based on the current QISKit transpilation methods. Each result is a transpilation pipeline fully compatible with a QISKit program. In addition, the training data can take into account the real specifications of real quantum computers.

Chapter 5

Automated Circuit Design

5.1 Introduction

In our first chapter, we look at automating the generation of quantum hardware. We start by looking at the problem space and identifying an appropriate method to perform the generation. Specifically, we use Genetic Improvement (GI) to target a graphical representation of a quantum circuit to reconfigure it to better run on different hardware. We construct an implementation of this method to test the accuracy of our predictions.

The most obvious first step was looking at the current non-automated approaches to circuit design. Applying automated approaches is one of the major focuses of the total thesis. Therefore, the most obvious first question is what is required for us to implement automation in the current landscape.

The purpose of this research is to identify the requirements of automation and implement these requirements in a way that shields a user from the complexity. From the background survey, we have identified that quantum computing has problems with the complexity of development. This leads to problems with methods being difficult, requiring specific expertise and hand-tuning which slows them down. We, therefore, look at mitigating the complexity by automating components of the development.

In this chapter, we demonstrate how to use GI as an automated and more generalized search approach to compiling quantum programs. It is used to configure an inputted program so that it runs with less error on particular hardware. This approach allows the optimization and development of quantum circuits without the need to hand-build or hand-tune final circuits. We demonstrate this by building circuits targeting multiple differing hardware backends. This is analogous to classical cross-compilation, where compiled programs developed for one hardware and operating system platform are transformed to work on another.

Two main areas of testing were performed. The first looked at random circuits built around parameters derived from real quantum machines. The second looked at pre-written quantum programs that targeted specific quantum hardware. The first set of tests showed that GI-compilations provided significant increases to the performance metrics of the circuits. The second set of tests showed that these performances were maintained when ported to real quantum circuits.

Based on our results, it is clear that GI is a viable method of automating the synthesis of quantum circuits. Our tests show two practical areas where GI provides tangible advantages. The first show an optimisation method that measurably lowers the error of the system. The result of this improvement increased the chance of any single run resulting in an accurate measurement, resulting in fewer repetitions needed to provide the same certainty in results. Repetitions mean more time spent on a computation which means that lowering error directly reduces computational time to run a circuit. The second tangible advantage is the ability to easily optimize circuits without an understanding of the background hardware. These advantages can be achieved with an input of a quantum circuit and the specifications of a quantum backend and do not require any field-dependent inputs. This allows a standard non-specialist would be able to run this with currently publicly available specifications.

The most important takeaway is that the GI refactoring method is both automated and does not require precise knowledge of the backend. In addition, it has results competitive with the state-of-the-art methods. **The impact of this is that development cycles taking advantage of this approach can achieve similar or improved performance while working with an overall simpler system.**

5.2 Background

Circuit Synthesis as a GI Problem

In order to run a program written for a quantum computer a compiler is required to translate human-readable code into operations that can be run physically on quantum hardware [43], [88], [44]. This is a multi-step process, normally consisting of first converting the higher-level description into a set of single operations that can be physically run on a real device. These descriptions are built using bespoke programming languages designed to target quantum machines. Such languages are generally referred to as quantum languages. The problem of circuit synthesis is the problem of translating descriptions written in such higher-level languages into circuits [31]. It is important to note that circuit synthesis is also an optimization problem. It is not sufficient to simply find any solution, as certain orderings or choices of gates will have preferred properties, such as shorter run-time or lower amount of error.

It is challenging to exactly synthesise an arbitrary circuit into physical quantum hardware. It is, therefore, necessary to rely on heuristics in areas where the problem size is too large to be efficiently solvable. These heuristics include search-based methods. As these approaches are dealing with large search spaces, the search method must be able to search cleverly by considering the structure of the problem which requires insight into the structure of the circuit. GI has long been suggested as a method for approaching the quantum synthesis problem in an automated fashion [79]. Methods have found a reasonable degree of success already [74].

GI is a search method already used for program synthesis which takes existing programs and uses automated search to generate better ones. GI has been used previously to modify the software to better run on different hardware [52]. In addition, modern quantum computers have several common metrics to assess their quality, such as total error or length of operation. The existence of clear metrics is useful when applying a GI method, where representation and assessment of solutions can be an early roadblock.

Flexible Representations of Quantum Circuits Using ZX

In order to apply GI techniques, it is necessary to find a representation of quantum circuits that can be flexibly iterated without changing the functionality of the program. This is further complicated as quantum circuits already have difficulties with efficient representations [91]. The inherent parallelism of a quantum circuit means that standard numeric methods of representation can be hard for humans to follow or even parse. This can be seen in matrix representations, which become unwieldy with even a relatively small set of qubits. It is therefore more common for quantum systems to be described using graphical or diagrammatic methods, which provide larger and more readable displays of functionality. While this solves the readability issue, almost all graphical notations are much less flexible than other descriptions. Most lack generalizable rules (called rewrite rules) to replace the structure of the graph with a different but equivalent structure (for example: [98] [70] [45]). Without generalizable rules automating changes very difficult.

The graph-based description of quantum circuits called ZX (discussed in 2.2.5 is a notable exception to this. It has been designed as a graphical calculus with an inbuilt core of generalized rewrite rules [21]. This fixes one of the major issues of other graphical quantum notations. The rewrite rules maintain the functional equivalency of a system but do not maintain an identical diagram. Here we recall that as equal but non-identical diagrams are moved through layers of a software stack they produce equal but still non-identical lower-level descriptions. This additional flexibility is often overlooked in a field that mostly relies on bespoke designs for very specific circuits. However, as quantum computing develops it is becoming increasingly necessary to improve the capacity to generalize. This is useful both in translating between different competing standards as well as bringing older approaches up to date with new circuitry

standards.

IBM-Q

Empirical results must be grounded in practicality. We, therefore, introduce IBM's Quantum Experience (IBM-Q) [25] as a set of several simple quantum computers made publicly available [5] by IBM to be used as the main benchmark of our tests. These are two five-qubit computers, two sixteen-qubit computers, and a single twenty-qubit computer. These are made available with a cloud control structure. This means that quantum algorithms can be run remotely. Data about the current run-time of the program is available through a function, allowing the number of registers, run-time and step count to be looked at. This provides sufficient metrics to be able to compare the performance of different algorithms.

Full details of the hardware and software used are publicly available. Furthermore, complete tests can be run on both the physical computer and a simulator. This generates a clear comparison of both the theoretical and actual computations. Practical sets of tests can be implemented to empirically show any theoretical improvements. Currently, there exist several standard algorithms, including a set of simple benchmarking tests for quantum computers. While the IBM-Q machines are relatively small, they are designed using standard principles for quantum machines.

5.3 The Structure of a Quantum Program

Due to greater limitations on how quantum programs are physically implemented, it is necessary to look at a more holistic view of their development. Quantum programs initially start as mathematical representations of a function, before being translated into more specific implementations until eventually a direct specification of a circuit exists. This is referred to as a software toolchain in classical machines. This is an important consideration as optimisation, and therefore the application of a GI technique as well, will have different constraints and advantages at each stage.

Let us consider an idealized quantum toolchain [35]. The quantum stack starts with a high-level description of the algorithm. The high-level description is typically the human inputted section of the stack. It is then broken down into its source code to generate a list of instructions to be completed. The instructions are then mapped to a low-level physical layout map which can be run on a physical circuit.

Overall, the algorithm exists in five states across their compilation: high-level language, low-level set of instructions, a description of the circuit, layout map and physical hardware.

As an example, we can look at the toolchain used to run programs on the IBM-Q machines.

We start with a quantum algorithm:

Algorithm 1 Simple Quantum Algorithm

Apply a Hadamard Gate to qubit 1
if Qubit 1 is $|1\rangle$ **then**
  Apply a not gate to qubit 2
end if

We then convert this to QISKit, a high-level quantum language.

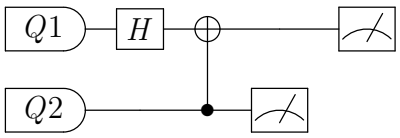
```
qc.h(qr[1])
qc.cx(qr[1],qr[0])
qc.measure(qr[0],cr[0])
qc.measure(qr[1],cr[1])
```

This can be broken down into an instruction set, called QASM.

```
qreg q[5];
creg c[5];

h q[0];
cx q[0],q[1];
measure q[1] -> c[1]
measure q[0] -> c[0]
```

The instruction set can be used to generate the following unique circuit map.



With such a small circuit it is trivial to find a functional assignment. We only require a single two-qubit gate, so we can assign the first and second qubit in our QASM description to the first (0) and second (1) qubit in our hardware model. With these assignments, the circuit can then be run on that backend. With each additional configuration of two-qubit gates, these assignments become increasingly difficult.

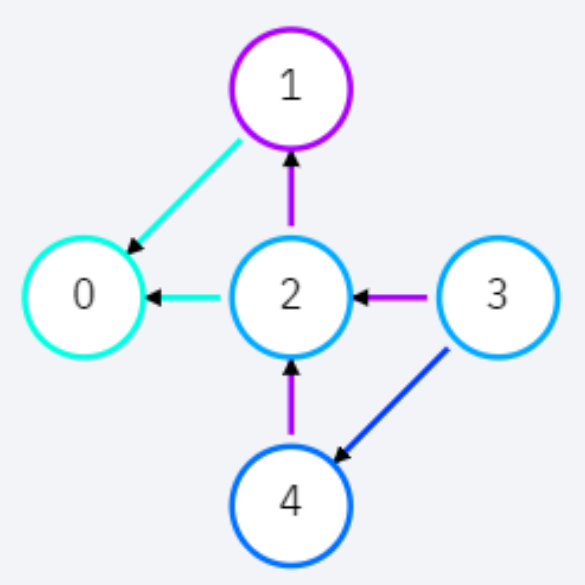


Figure 5.1: *A diagram showing the locations and connections in an example quantum computer. Error is included with the data by a spectrum from cyan (low error) to purple (high error). Errors for single qubit operations are represented on nodes, while errors for two qubit operations are represented as lines.*

5.4 Method

We start with a summary of our method. We take a hardware model of a quantum computer. This is our target backend. We also take a quantum circuit. As discussed, the intent is to implement the circuit on the target backend. We convert the quantum circuit to a ZX-graph. This ZX-graph is duplicated to form a population. Each individual is mutated where possible randomly with a set of available rewrite rules. The graph representing the lowest error is then selected and the process repeats until no way of applying rewrite rules can be applied or the maximum number of repetitions is met. This graph is then returned to a circuit which is assessed to provide the final measure of the quality of the compilation.

5.4.1 The Hardware Model

The hardware model in figure 5.1 shows the structure of one of our example quantum computers. The example models are built off a reference of real IBM machines that have been made available during our research. There are two important pieces of information we take from this model. First, we want to know the average error for implementing a gate on a specific qubit. Secondly, we want to know the error for implementing a two-qubit gate across a pair of qubits. We refer to low-error (cyan), medium-error (blue), or high-error (purple) for the purposes of discussion, but attach

the exact error rate in our model.

It is sometimes necessary to implement two-qubit gates that cannot be run directly onto the qubits they are assigned to. While this is possible, it is also an expensive step. The current automated approach relies on applying additional SWAP gates until the gate can be run (because the gate requires physical adjacency of the qubits it works on). The exact error of this depends on the circuit structure and cannot be exactly calculated, it is clearly an expensive approach. An estimate was generated by benchmarking a selection of circuits. On average, the cost of implementing a single such difficult gate can be considered to be roughly four times the cost of a two-qubit gate and therefore a huge source of gate error. We refer to these gates as *hardware-adverse gates*, as the cost of these gates isn't their inherent complexity but that the physical architecture cannot easily implement them.

5.4.2 Fitness Function

The quality of a quantum circuit is difficult to assess. In addition to classical measurements of costs, such as run-time or resource use, there are additional factors introduced by quantum computers being in the early stages of development. One of the most important of these comes about from the inherently probabilistic nature of quantum computers. Any single run of a code cannot measure a quantum state, even without noise. Instead, the same circuit must be repeated many times. This is further complicated by quantum computer's high levels of noise. Further repetition can counteract this noise but higher levels will require more repetitions, often to a quite drastic degree.

For this reason, one of the most common measures of quality of a quantum program is the total error rate of running the circuit. This error rate is built up from a number of things, including hardware and background noise. From a circuit synthesis perspective, the most relevant cause of the error is gate error. **We, therefore, use gate error rates as our fitness function.** Each individual error contributes a probability of an individual failure occurring. The total error rate is then calculated by looking at the combined probability of any error happening (i.e. the run not being error-less).

5.4.3 Description of the Individual

To apply a genetic algorithm we must select an appropriate representation of the problem. This representation is difficult with quantum computers. Many methods use graphical representations of circuits. However, most of these graphical representations lack generic semantic-preserving rewrite rules. Without generic rewrite rules, applying mutation operators on arbitrary circuits can be difficult.

A notable exception to this is the language ZX. ZX graphs have rewrite rules that maintain semantic equivalency between circuits. This lets us mutate our ZX-graphs.

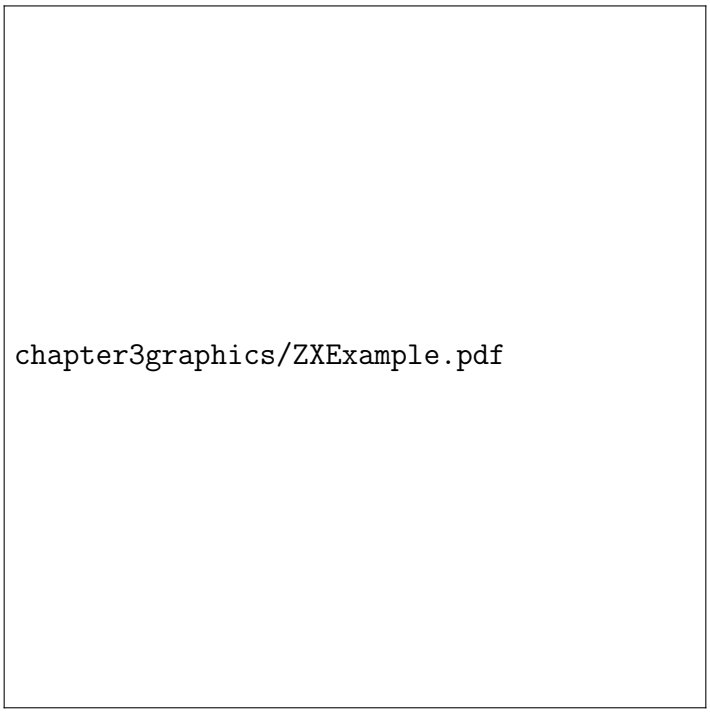


Figure 5.2: *A simple example of a ZX-graph, demonstrating a Hadamard gate as a blue line and two connected nodes forming a controlled not structure.*

Each individual consists of a single graph described with the ZX-calculus. The original population consists solely of copies of the original circuit. The PyZX package [49] provides a datatype for manipulating ZX-graphs in python. This contains a list of labelled gates and their output and input connections.

5.4.4 Evolutionary Process

At each iteration, a random set of rules are applied to individuals to rewrite them into equivalent but differently structured circuits. These rewrite rules are called mutations. The new mutated individuals are combined with the original population to generate the new population. The strategy starts with a high probability of each mutation being applied, but this decreases over generations. This allows us to perform a more efficient, structured search of the total problem space.

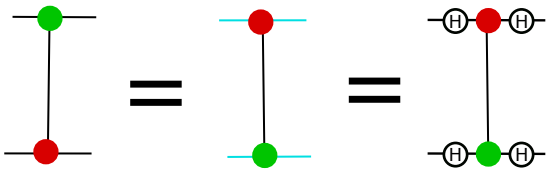
The following rewrite rules were used as mutations:

Identity Removal

An identity set is a set of gates that in combination do not change the system. Identity removal is deleting these sets. Alternatively, the rule can be reversed by introducing a combination of gates that total to an identity. The benefit of adding in additional gates is not immediately obvious but additional gates can provide flexibility in how gates are formed. This can increase both the speed of the search as well as improving the final result.

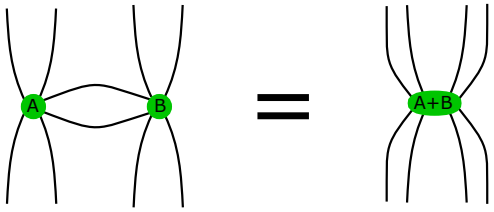
CNOT Reversal

CNOT gates are two-qubit operations with a direction from one qubit to the other. Pairs of qubits from the quantum computers we used could only implement one of the directions. CNOT reversal changes the direction of the operation. This has a tangible effect on the system and requires the addition of supplementary gates to compensate.



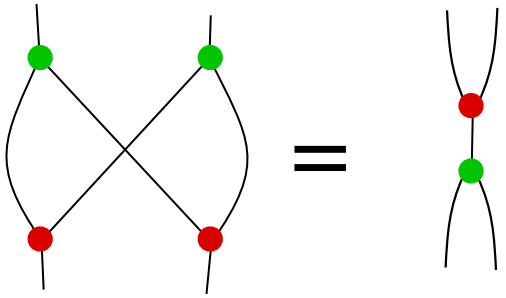
Spider Fusion

A spider fusion, or node fusion, aims to combine two nodes into a single one. Nodes represent qubits, so combining two nodes is equivalent to deciding that operations being split across two qubits can all be performed on a single qubit instead. This requires that both the operations and the computational basis are compatible to be performed. The reverse is also possible, by separating a single node into two different nodes.



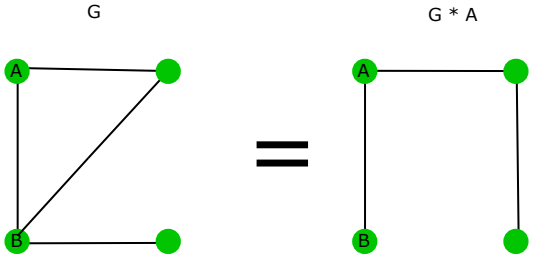
Bialgebra Equation

The bialgebra equation simplifies areas where two qubits both use each other as inputs in the operations being applied to them. The rewrite rule simplifies the graph cycle (called a 2-cycle) into a single combined set of operations. This can be used to rewrite it as a set of operations on a single qubit with a combined state, rather than two different ones.



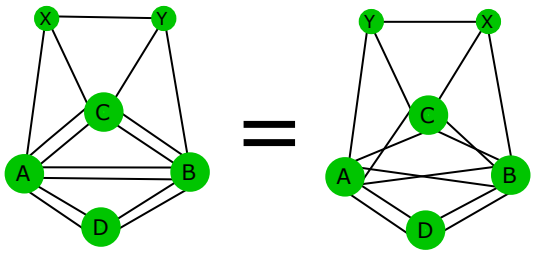
Local complementation

To create the local complementation of a graph, we take a target node and then form a set of all its neighbours. For each pair of neighbours, if they are unconnected then they become connected and vice versa. Reapplying this to the same node will reverse the process. This ends up performing the same function but changes the pathway the operation takes across qubits.



Pivoting

Pivoting is the act of exchanging the position of two connected nodes on a graph. In order to maintain the equivalence of the graph, the neighbour nodes of the two nodes are locally complemented. Reapplying this to the same pair will reverse the process. This allows the system to swap the position of two qubits while performing the same functions.



Selection Process

This now provides the following algorithm to generate circuits.

Algorithm 2 GI Circuit Search

```
Initialize parent and evaluate it
while Max generations not reached do
    Select highest evaluated circuit from population
    while Best circuit has mutations that can be applied to it do
        Apply a random set of mutations
        Evaluate the mutated offspring and add it to the population
    end while
end while
Select highest evaluated circuit
```

5.4.5 Test Setup

The aim of our experiments is to show that our GI method can perform competitively with standard methods. We also aim to show that this method can perform tasks that would normally require hand-tuning, such as targeting the output circuit at particular hardware.

Our results focus on three machines. These are two five-qubit machines, called Yorktown and Tenerife, as well as a single fourteen-qubit machine, Melbourne.

Our first set of tests is looks at random circuits. The initial circuits include a specific amount of qubits and gates. We do not restrict the gate count of the optimized circuits. However, it is enforced that the number of qubits must be within the bounds of the machine. These are not likely to represent real algorithms. However, they are likely to represent realistic combinations of gates that would appear in real algorithms, providing a useful benchmark. For the five-qubit machines, we generate test circuits from each pair of 2, 3, 4, or 5 qubits and 3, 5, 10, 20, 30, 50 or 100 gates. For

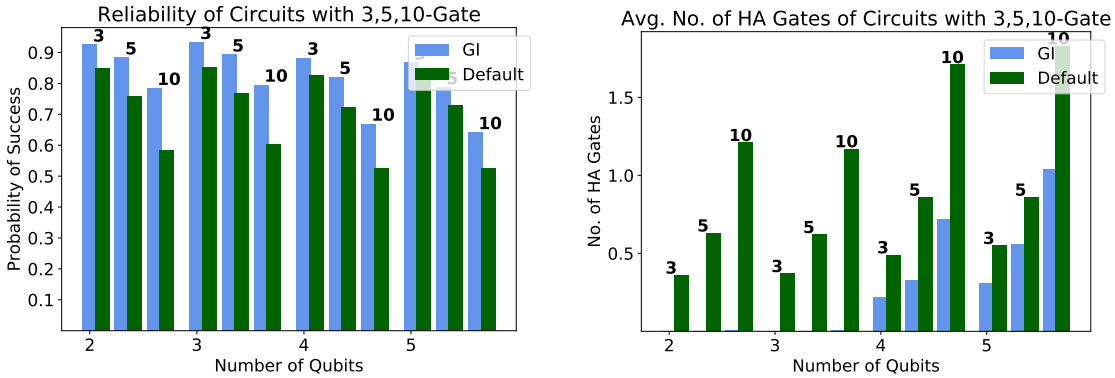


Figure 5.3: *Bar graphs showing the likelihood of success of a single run and the average number of hardware-adverse gates removed by our GI method and a standard search method applied to the small circuits on the Tenerife hardware*

14 qubit machines, we extend our tests to look at circuits with 10,11,12,13 or 14 qubits.

We have run a selection of programs written in QASM to turn into circuits. These were taken from a now unavailable QASM github page of low-level examples. If these experiments were to be repeated, the recommended alternative would be the open-QASM examples page [66].

5.4.6 Results

We compare our GI approach to simple search methods. We first compare the likelihood that the gate error will result in a bit-flip in a single run of the circuit. We then look at the number of hardware-adverse gates removed. Each plotted line represents the average of 20 different circuits.

As expected, the GI approach results in a higher likelihood of success (Fig 5.3 and 5.4 left-side) and lowers the number of hardware-adverse gates (Fig 5.3 and 5.4 right-side). We performed a one-sided Mann–Whitney U test to confirm where the GI approach outperformed. We report a match as statistically significant if the p-values of the test was below 0.05.

Our results show that the GI method results in a compilation with a higher likelihood of success and fewer hardware-adverse gates in both Tenerife (Fig 5.3) and Yorktown (Fig 5.4) for small numbers of gates. Of particular note is that for small numbers of gates and qubits, hardware-averse gates were completely removed while the alternative occasionally missed one.

This trend of better performance of the GI continued on the circuits with larger gate counts (Fig 5.5 and 5.6). Note the circuit containing one hundred gates is intentionally designed to stretch the system. While the success rate remains low, there is a significant

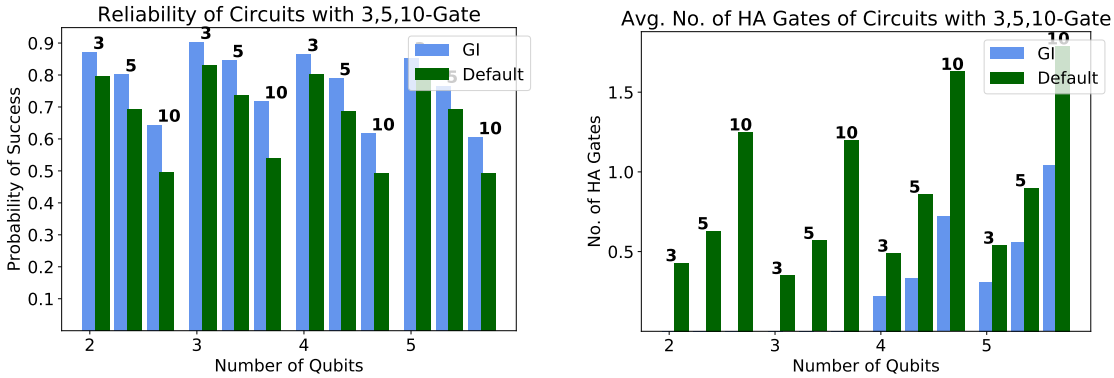


Figure 5.4: *Bar graphs showing the likelihood of success of a single run and the average number of hardware-adverse gates removed by our GI method and a standard search method applied to the small circuits on the Yorktown hardware*

increase in the likelihood of success. Even in the worst case, **around a fifth of all problematic gates were removed**.

The larger search space of Melbourne combined with the more rigid structure makes improvement much more difficult (Figs 5.7 and 5.8). It is also notable that at small gate-counts (5 and below), tests generally failed to achieve statistically significant improvement. The Melbourne hardware is designed in a much different way to both Yorktown and Tenerife. The qubits are arranged in a long chain, with a relatively rigid structure of connections. This makes optimization comparatively harder.

However, even with this, the improvements were large enough to have a significant impact. At 50 gates (enough for a reasonable program), the average amount of error reduced was 5%. While this does not seem large, the difference between a 10% chance of success and a 14% means a difference of 167 and 85 repetitions required for a 99% confidence in a correct answer. **This is almost half the run time.**

As a further interesting point, the error rate of each gate is in general lower than on the other two devices without considering hardware constraints. This makes Melbourne the best backend in the purely theoretical case, before considering the hardware limitations. While in practice, compilation was most difficult on this machine. This is useful as an example of how knowledge of specifics of the hardware provides a distinct compilation advantage.

We finally take the previously mentioned set of algorithms written in QASM as examples of real quantum code. These represent simple, ubiquitous algorithms used as examples. We aim to show that these algorithms can be rebuilt to run on different backends, something not otherwise possible with QASM code.

The results of testing these algorithms (Fig 5.9) show a major success for our

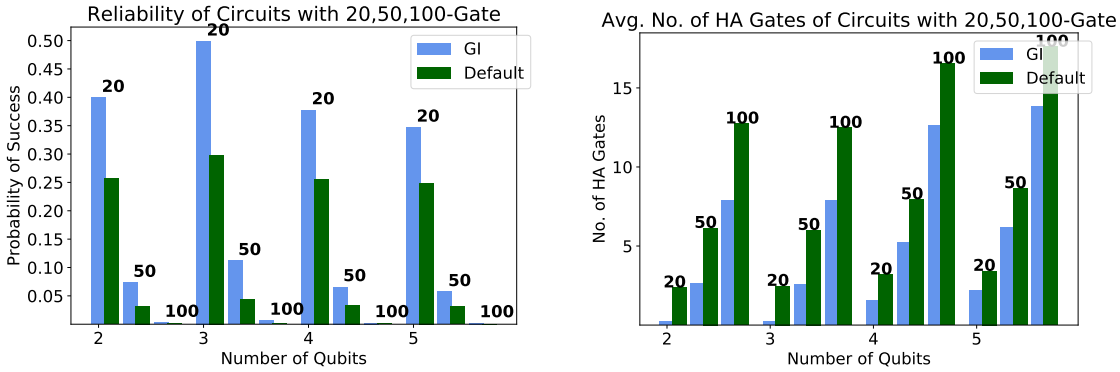


Figure 5.5: *Bar graphs showing the likelihood of success of a single run and the average number of hardware-adverse gates removed by our GI method and a standard search method applied to the large circuits on the Yorktown hardware*

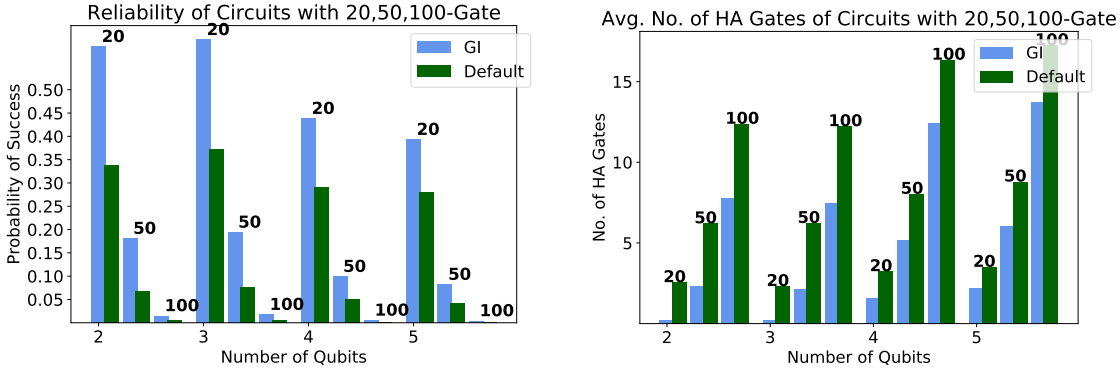


Figure 5.6: *Bar graphs showing the likelihood of success of a single run accomplished by the best circuit resulting from running our GI method and a standard search method applied to the large circuits on the Tenerife hardware*

methodology. We were able to fully implement four out of six of the algorithm on at least one backend.

5.5 Chapter Summary

In this chapter, we have demonstrated a method of applying a Genetic Improvement method to quantum circuit synthesis. We start by parsing QISKit implementations of quantum circuits and translating these into ZX-graph representations. This graph representation allows more generic rewrite rules to be applied to the graph. These rewrite rules can be applied to change the structure and potentially the properties of the circuit without changing the functionality. By applying these rewrite rules as

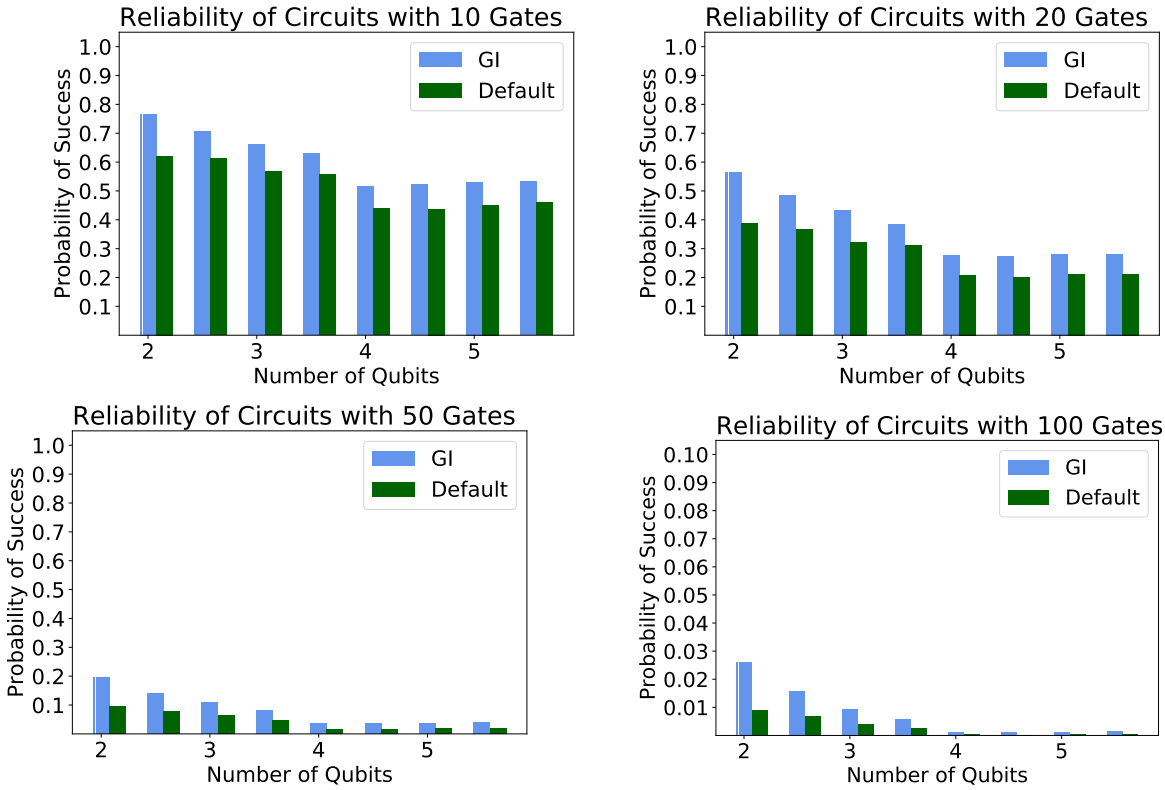


Figure 5.7: *Bar graphs showing the likelihood of success of a single run accomplished by the best circuit resulting from running our GI method and a standard search method applied to a selection of various-sized circuits on the Melbourne hardware*

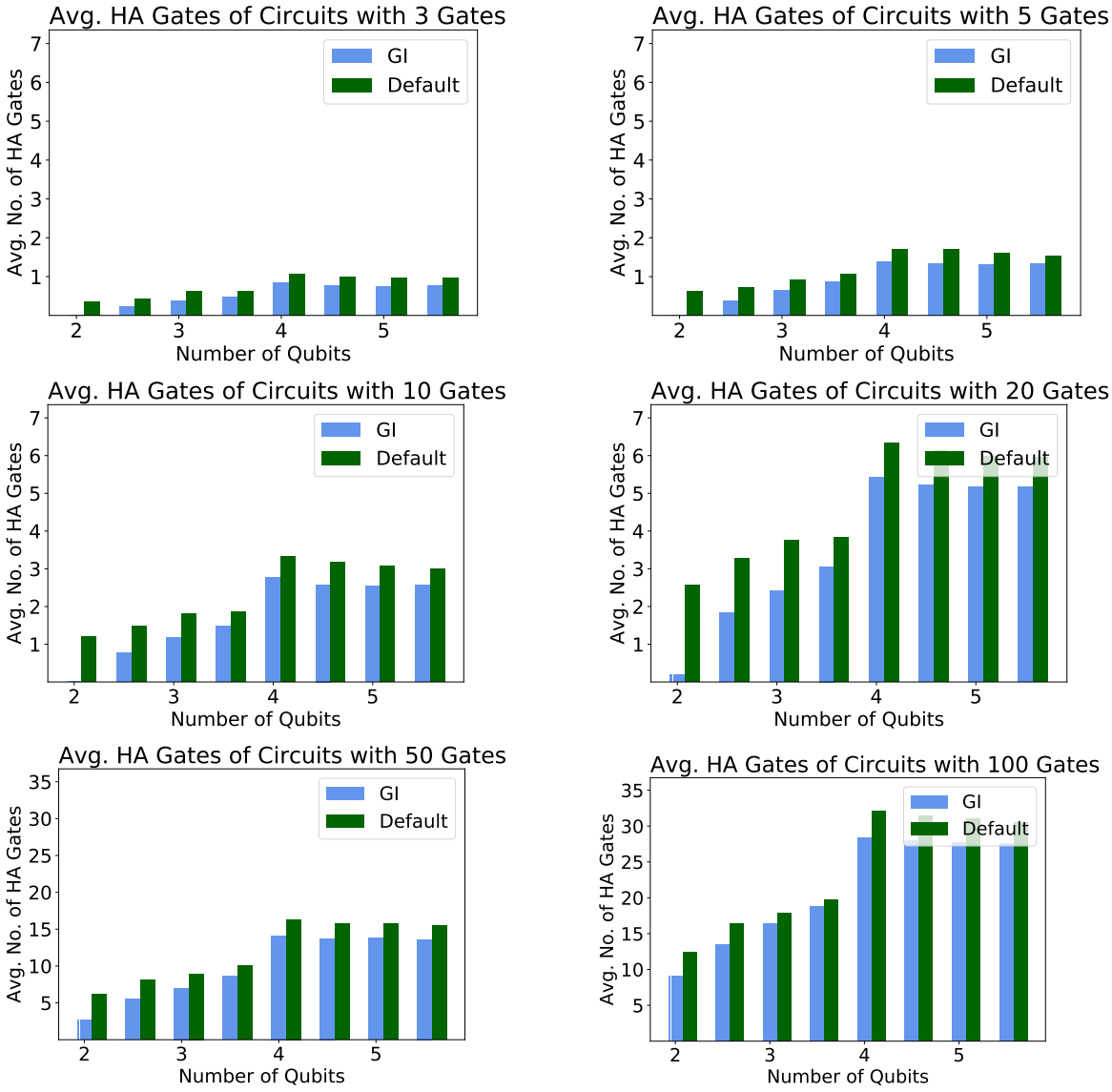


Figure 5.8: *Bar graphs showing the average number of hardware-adverse gates removed by our GI method and a standard search method applied to a selection of various-sized circuits on the Melbourne hardware.*

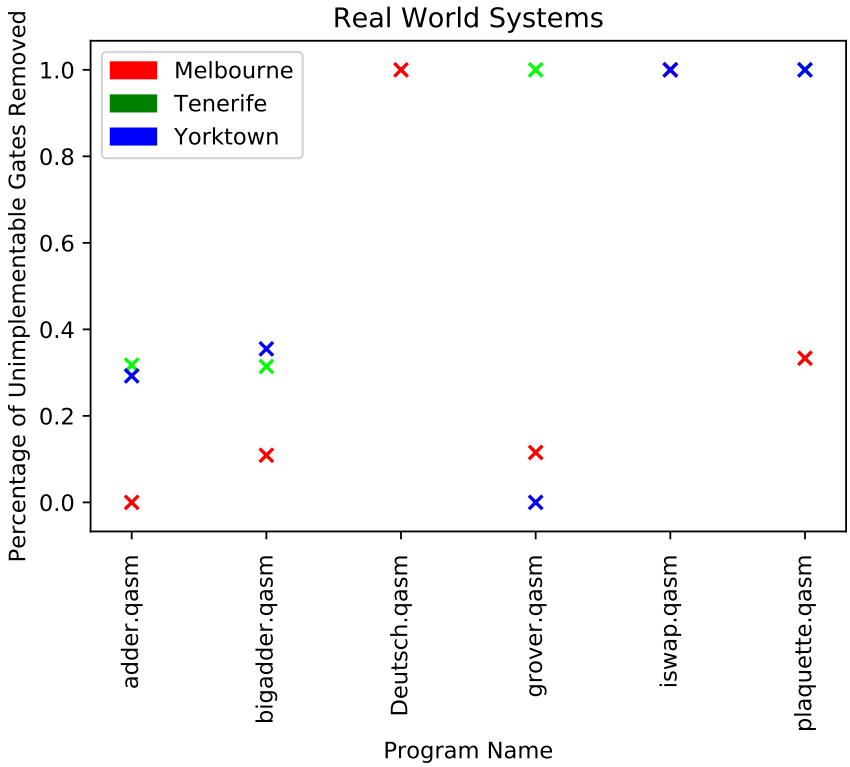


Figure 5.9: *A bar graph showing the percentage of unimplementable gates removed from practical examples of quantum code.*

mutation functions on a population on ZX-graphs it is possible to efficiently search through the problem space. We implemented this method in python.

We demonstrate a set of practical results based on this method. These results provide evidence of the fact that it is possible to improve circuit synthesis by providing better models of hardware back-ends and integrating this into the compilation. We measure this improvement by looking at the chance of any single run resulting in an accurate measurement. The results show an improvement in this metric can be made by applying our method. This means that when running our method, **fewer repetitions are required to get the same degree of accuracy.** This means that less computational resource is required for the same problem. This is particularly notable in the tests generated from real example problems. The data showed significant improvement. We expect that use of a Genetic Improvement method like ours would provide a significant boost to the speed and ease of development of new algorithms on new quantum hardware and therefore help speed up quantum research as a whole.

Chapter 6

Automated Scheduling of Qubits

6.1 Introduction

6.1.1 The Advantages of Automated Scheduling

In this chapter, we aim to use automation to improve the speed and ease of quantum development. The aim is to remove human input from a section of the quantum program synthesis process while maintaining the advantages of the design. The method of automation we use is building a (PDDL) scheduler to turn abstract gate instructions into an ordered schedule of actions that a physical machine can take. Since no code needs to be written or parameters set, this means this process can be initiated with minimal human time cost and expertise. In addition, the scheduler produces programs that are competitive, or superior, in quality to current methods.

Lowering the difficulty of quantum compilation is useful because of the simple fact that designing quantum algorithms and computers is very hard [1]. While qubits provide a powerful core for quantum computers, the surrounding infrastructure is still lacking. The current hardware is limited by how qubits are connected and how they can be interacted with. This makes synthesising programs hard as it requires exploiting as much computation power as possible while working around the complex limitations and interactions of the hardware [69]. One of the stronger proposals for getting around the difficulty was Lee Spector's work on automating development through Genetic Programming [79]. This proposal includes several other suggestions for possible methods of automation any of which may prove useful.

The main challenge preventing the automation of quantum computers is implementing the surrounding structure that would facilitate them. While current methods are improving on structure, we are still a long way off matching classical approaches. In addition, quantum computing is less developed and less intuitive than other classical methods. To get a full understanding of a quantum system, would require a huge amount of expertise that isn't feasible for every developer to have. By structuring the

software development process cleverly it can be broken down into smaller, more manageable steps. These can then be solved by humans with a less complete understanding of the systems.

In this chapter, we break down the generation of a low-level hardware instruction set from high-level code into small, modular steps and automate them. We use the Planning Domain Definition Language (PDDL) to model each circuit synthesis problem in a way that is compatible with standard AI planning techniques. We use this method to automatically optimize code for specific hardware; this is a task that would normally require human intervention.

We extend previous work [90] on modelling realistic hardware by taking into account that not all qubits are equal. Qubits in different positions may be more likely to suffer errors, have a shorter lifespan or have more limited connections to other qubits. We demonstrate that a preference for better qubits provides a better final result. Our objective is to provide a set of empirical tests that show this being applied to real algorithms that can be run on real quantum hardware.

6.1.2 Contributions

- Demonstrate a PDDL model for the automated scheduling hardware use and gate ordering on quantum computers
- Present this method in the context of a modular software stack structure of program development that reduces the user expertise required
- Apply this method to real and generated quantum circuits, showing that the automated method provides benefits to larger circuits where hand-tuning is infeasible

6.1.3 Compiling Quantum Algorithms with a Software Stack

Quantum computers cannot run complex unitary transformations in a single step. In practical quantum algorithms, larger unitary transformations are split into smaller physically implementable steps. These now provide a model of a low-level quantum circuit. These circuits are currently specified on idealized hardware, which ignores the details that are specific to actual types of hardware. While this is helpful to programmers who are building the program, it does mean that they cannot be directly run on any given quantum computer, so a further breakdown is required.

It is therefore necessary to look at how a high-level algorithm description is translated into a set of operations that can be physically run by a quantum machine. This is known as quantum circuit compilation (QCC). Compiling a quantum circuit means

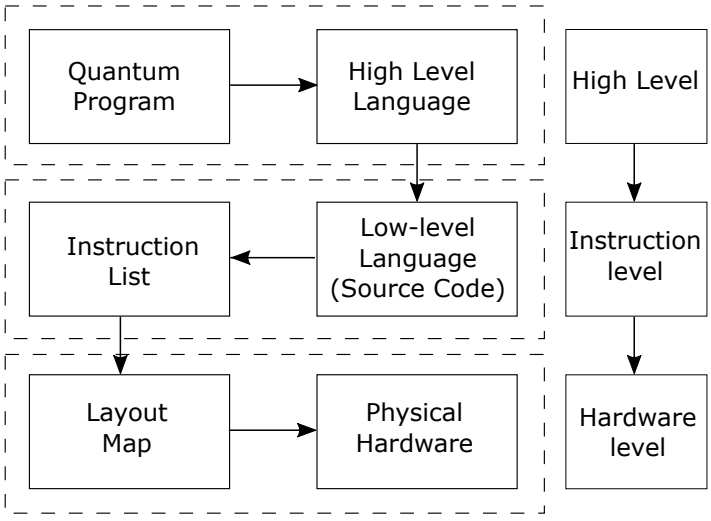


Figure 6.1: *The Quantum Compilation Software Stack. The left hand side shows the pathway through stages while the right hand side shows the abstraction level of the stages.*

reorganizing or adding gates to ensure that all operations are physically implementable within the limitations of the specific quantum hardware. This process is often done in several steps, moving through layers such as completely abstract definitions, human-readable code and machine instructions before being translated into physical operations. A software stack is simply a map of these layers as well as the techniques and tools used to move across these layers.

We look at a description of an idealized quantum toolchain [35]. The quantum stack starts with a high-level description of the algorithm. The high-level description is typically the human inputted section of the stack. This description is then broken down into its source code to generate a list of instructions to be completed. The instructions are then mapped to a low-level physical layout map which can be run on a physical circuit. Overall, the algorithm exists in five states across their compilation: high-level language, low-level source code, derived set instructions, circuit layout map and physical hardware.

This process is complicated and difficult. Generating a useful circuit that matches a high-level specification of several steps that require specific domain knowledge. This can be manual data-allocation at higher levels or even specific hardware knowledge at the very low levels. At present, even the most accessible compilation methods have some kind of hand-tuning integrated into them. We believe that automated software engineering methods are an essential tool to reduce the complexity of these systems. Automated program synthesis allows the generation of circuits that automatically take into account the specifics of a quantum system while reducing how much the user is

required to know themselves.

We emphasise that while there is a large focus on looking at this as a satisfaction problem, it is also an optimization problem. Multiple orderings of gates can perform the same function. However, different orderings can result in different operations being performed in different places. This change in positioning allows the system to better organize resources to run efficiently and to prioritize qubits that can run gates with less chance of error. This results in a circuit that can run faster or have better odds of a successful run.

In this chapter, we present a method of automating quantum program synthesis. This method looks at treating the scheduling of gates on qubits as a planning problem with an explicit time-state. The method provides an advantage over other methods by providing an automation approach that does not require an input of human expertise about the quantum system. In addition, we consider the practical error introduced by running the produced circuits on real machine hardware. This produces a final product that better aligns with the target hardware.

6.1.4 QAOA

The quantum-approximate-optimization-algorithm (QAOA) is a quantum algorithm that looks at generating an approximate solution of NP-hard problems. [92] QAOA is a large focus of quantum computing as it shows promise of achieving quantum supremacy in an area with real-world applications. [84] QAOA aims to solve combinatorial problems where a criterion must be maximised by encoding them in such a way that the value to be maximised becomes equal to the energy level of a quantum system. The lowest level of energy possible in the system (called the ground state) becomes the optimum solution to the problem. For a heuristic approach, this means that we can find the best solution by taking the result that has the lowest measured energy level.

An example of this is the maxcut problem. The maxcut problem takes a graph of nodes connected by edges and wants to split it into two sets such that the maximum amount of edges are connected to nodes in both sets. This is useful in areas such as large-scale circuit design, database clustering or any other field where the amount of communication channels needs to be as large as possible.

QAOA is a popular area of focus for a number of reasons. First of all, the algorithm contains large sets of commuting gates. This makes it easier to reorder the operations and provides more flexibility for optimization. Secondly, the circuit depth of QAOA is relatively small allowing it to be run with the currently limited coherency times of modern devices. Finally, approximation methods are more robust against the inherently probabilistic nature of a quantum machine. For these reasons, QAOA represents the current state-of-the-art of quantum circuit synthesis problems and will be used as

a baseline objective for our methodology.

6.1.5 Temporal Planning

Automated planning is a branch of Artificial Intelligence that aims to solve specific objectives by selecting an ordered series of actions to perform. [56] These problems are defined by a description of the initial state, a set of end goals and the set of actions that can be performed. The simplest case of these is called the Classical Planning Problem. These problems require a single known initial state, actions that are instantaneous and deterministic, and only a single track to perform actions.

Temporal Planning Problems are an extension of Classical Problems that allow the expression of more complicated problems. [9] Specifically, Temporal Planning introduces the idea of a meaningful time state by providing actions with a duration and specific time to occur. It is also possible to directly specify what resources are used when over the duration of an action. With a better model of resource availability, it is possible to allow multiple actions simultaneously, provided the resources are available at the time.

6.1.6 PDDL

PDDL (Planning Domain Definition Language) is a formalism that aims to standardize AI planning problem descriptions [36]. PDDL provides a modular approach to describing planners, problem domains and individual problems as separate models. Domains and problems are given to planner programs that attempt to solve them. PDDL problem domains describe the larger context of the problem. Domain definitions consist, at minimum, of basic predicates and operators. They may also commonly contain definitions of typing, constants, and functions. The other part is the problem definition. This describes the specifics of a single specific problem. This includes exact values for the initial state, objects present and the goal state. Multiple problem definitions can be written for the same domain.

PDDL is overwhelmingly the most popular planning domain language which helps to capitalise on its modular nature. As a popular planning domain, a wide range of planners have been developed in PDDL. Because of the modular design, if improvements are made to one stage of the problem-solving process then they can be easily integrated with the rest of the process. In addition, any new developments can replace older systems very easily. Importantly, the current leading planning work comes from the International Planning Competition which uses PDDL. Overall, this gives a reasonable expectation that the best planner is likely to be a PDDL planner. In addition, were a better non-PDDL planner to exist, the ability to swap out planners is preferable to accessing a more powerful planner now, as our focus is on automating

to reduce complexity not only performance. This makes PDDL the obvious choice of language to access the largest range and state of the art planners.

6.1.7 Applying Planning to IBM-Q

In chapter 2.5 we introduced IBM's Quantum Experience as a set of several simple Quantum computers made publicly available by IBM via the cloud. It is useful to be able to run our code to help ground our theory in practical results. The advantage of using real, accessible machines is that our resulting circuits can be tested to ensure that the theoretical results have a basis in reality. The full details of the hardware and software used are publicly available, with multiple different machines (or backends) with different sizes and architectures [4]. Circuit maps, error rates and other information about the quantum hardware is also made available. In addition, the current runtime of the program is available through a function, allowing the number of registers, run-time and step count to be looked at. This provides sufficient metrics to be able to compare the performance of different circuits.

IBM-QE's current compilation method involves assigning instructions to qubits during runtime without planning. This requires that states that cannot be run in their current position to be moved through expensive swap functions between qubits. This can be particularly problematic when neighbouring qubits are already in use. In these cases, either inefficient paths need to be taken or the circuit simply needs to wait for another qubit to become available. As quantum computers states have short lifespans before error sets in, these time costs can be very detrimental. Planning techniques aim to improve this by looking ahead to avoid areas where these steps are necessary and ensuring that when swapping is performed by necessity that it is done as smoothly as possible.

6.2 Modeling Quantum Program Synthesis as an Automated Scheduling Problem

6.2.1 A PDDL Software Stack

We start by presenting a structured stack to generate our qubit schedule.

The important inputs are the hardware and software specifications. These are the areas expected to be human-generated. These are parsed in an automated fashion into a single combined PDDL problem file. The PDDL domain is generated from a standard simple set of rewrite rules. The domain and problem file can then be fed into a PDDL planner. This planner then generates a schedule to run gates on qubits.

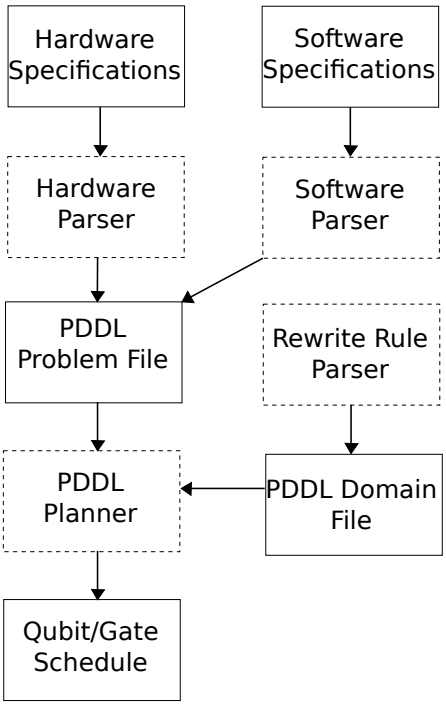


Figure 6.2: *The PDDL software stack. Hardware and software specifications are combined to make a PDDL problem file. This problem file is combined with a domain file in a planner to form a qubit schedule.*

6.2.2 The Domain

Our model keeps track of four separate systems. These are qubits, quantum states, pairings between two qubits and gates to be placed.

Qubits

Qubits refer to the physical qubits that make up the hardware. For qubits, we store a unique id, the error cost of a single gate on this qubit, whether or not the qubit is in use, and with which other qubits there exist a connecting gate of any sort.

Quantum States

Quantum states are the abstract representation of the state that is stored on a single qubit. With the application of a two-qubit swap gate, these can be moved between physical qubits. For each state, we store a unique ID and the current physical qubit that the state is located at.

Pairs

Pairings refer to the physical connections between two qubits. We store the IDs of the two qubits they connect and the cost of running a two-qubit gate through this connection.

Gates

For quantum gates, we store the ID of the quantum state or pair of states they need to be applied to, the length of time they require to run, and the set of other gates with which they commute.

In practice, this results in a pool of gates that need to be implemented on specific quantum states in a flexible order, and a definition of the physical system where they can be run.

6.2.3 Actions

An action in the system consists of scheduling the running of a single qubit gate on a qubit or a two-qubit gate on a pair of qubits. To schedule a gate on a qubit, that qubit needs to have no other gate scheduled on it at the same time and must contain the correct quantum state that the gate must be applied to. In addition, gates can be specified so that they must be performed before other gates.

The other main form of control is the application of swap gates. Assume two abstract qubits are represented (implemented) by two physical qubits, then a swap gate simply swaps the physical representations of those two physical qubits. The capacity to swap physical qubits gives us more flexibility when scheduling them.

6.2.4 The Problem

The problem domain defines both the physical hardware being targeted and the set of gates we wish to apply to it. The physical hardware consists of the qubits, the paired connections between the qubits and the costs and time-cost of applying a gate on each qubit/pair. Within the set of gates, each gate has attached to it both the quantum state it targets and the order of application. For ordering, each gate is grouped with other gates such that each gate's ordering in regards to the rest of the group does not impact the result computation (I.E. the gates that commute are grouped). The problem files were generated automatically by parsing QASM files and quantum hardware specification files.

The quantum circuit portion of the problems was generated with a combination of parameterised random generation and a selection of pre-written code, which is dis-

cussed in more detail further on.

The hardware specification portion of the problems was taken by parsing the details of four machines selected from the what was available on IBM-Q at the time of experimentation. These are three five-qubit machines, called Essex, London and Yorktown, as well as a single fourteen-qubit machine, Melbourne.

Essex and London both have the same structure, however, they have different quality of qubits in different locations. Yorktown is further differentiated by its different structure. Finally, Melbourne is a radically different backend with fourteen qubits with minimal connections.

6.3 Results

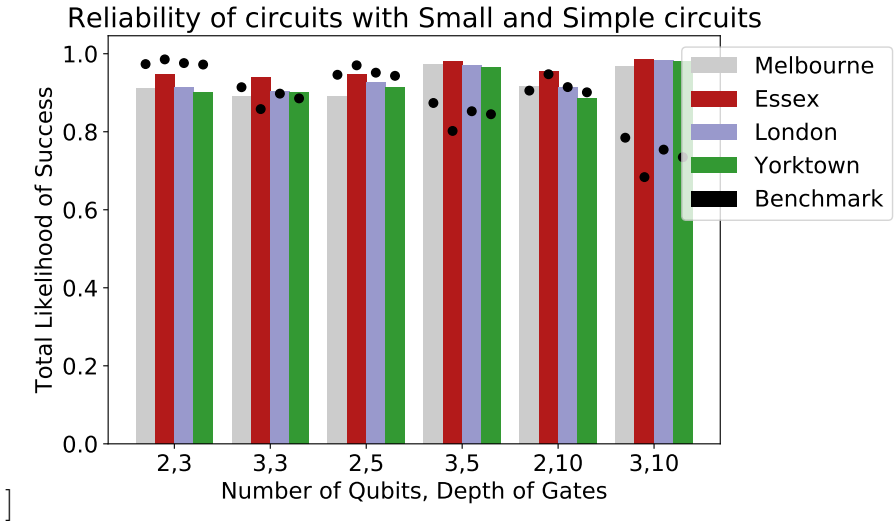
6.3.1 Structure of Experiments

Our first set of tests is built on optimizing randomly generated circuits. We generate random circuits that use a specific amount of qubits and gates. Randomized benchmarking [50] has a long history of a stable method of benchmarking quantum computer. The small size of current machines mean that random sets are likely to provide a reasonable approximation of structures expected to be found in quantum programs. This gives a good rough idea of quality even if it can't be used to predict or design specific circuits.

For the five-qubit machines, we generate test circuits from each pair of 2, 3, 4, or 5 qubits and 3, 5, 10, 20, 50 or 100 gates. We divide these into either simple circuits (2 or 3 qubits) or complex (4 or 5 qubits). Similarly, we separate the gate counts into small circuits where the number of gates is 10 or less and large circuits that have more than 10 gates. For each of these pairs, 20 individual circuits have been generated. For 14 qubit machines, we extend our tests to look at circuits with 13 or 14 qubits.

Our second set of tests looks at a set of simple algorithms. These include a wide range of different types of simple circuits and gate rewrites. This is to test the ability to compile integral cores of quantum computing, such as Toffoli gates and bell pairs. While initially done separately, these are combined into larger combination circuits.

As a benchmark, we used the default Qiskit "transpiler" function [86] which performs light optimization. The transpiler function is designed to compile code to a specific hardware. Here transpilation refers to a compilation of a quantum program between hardware models while maintaining the same level of abstraction. This function is built up by selecting a set of transpiler "passes" by hand to build a full circuit-to-hardware pipeline. Each pass performs a single portion of the transpilation, which can roughly be separated into layout selection, routing with swap gates, and optimiza-



]

Figure 6.3: *A bar graph showing the likelihood of success of a single run of each generated circuit for small amounts of qubits (2-3) and simple gate complexity (max gate depth of 3-10) on 4 types of hardware.*

tions/basis changes.

6.3.2 Randomized Circuits

We start our summary of results by looking at our randomly generated circuits.

Small, Simple Circuits

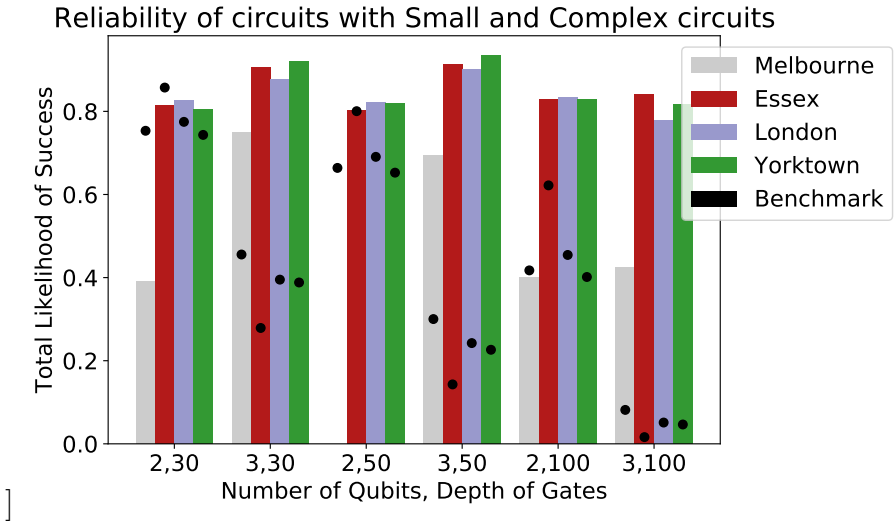
Our first results look at the lowest level of circuits, consisting of small numbers of qubits and low gate counts.

The smallest, simplest circuits use only two qubits and run with only three gates. The PDDL method performs a little worse than the benchmark. As the problem becomes more complex, the planner's results stay high. The increased flexibility of a larger system mitigates the increased problem size. In comparison, our benchmark shows diminishing results.

Small, Complex Circuits

We now show circuits with the same small qubit count, but larger gate depth.

At this stage, we maintain the small number of qubits but introduce a large number of gates to each circuit. This time, the number of gates involved in the two-qubit problems mean that the planner can build up an incremental advantage. At 30 gates, the results are similar but at 100 the planning method is clearly outperforming the



]

Figure 6.4: *A bar graph showing the likelihood of success of a single run of each generated circuit for small amounts of qubits (2-3) and larger gate complexity (max gate depth of 30-100) on 4 types of hardware.*

benchmark. With the three-qubit circuits, the planner performed consistently better.

Notably, the 16Melbourne machine proved difficult. It is a difficult backend to compile for without tailoring a method to it, as it only has a limited loop structure of connections between qubits. Despite this, the planner performed well on larger circuits.

Large, Simple Circuits

We now show circuits with larger qubit counts but small circuit depth.

The large simple circuits look at higher numbers of qubits with small amounts of gates. As before, the performance of the benchmark trended downwards as the size and complexity increased while the PDDL method was much less affected. Notably, the change from 4 to 5 qubits increased the quality of the PDDL compilations, rather than reducing them.

Large, Complex Circuits

Finally, we look at larger qubit counts and large gate depth.

These areas were designed to test the upper limits of the system. Many of the one hundred gate algorithms were unsuccessful, as were many of the Melbourne problems. It was expected that circuits with a maximum gate depth of 100 would not have been solvable by the planner at all. However, the planner managed to solve some of these very large problems. The five-qubit large circuits represent the largest success, resulting in fidelity high enough to be reasonably run on a real machine.

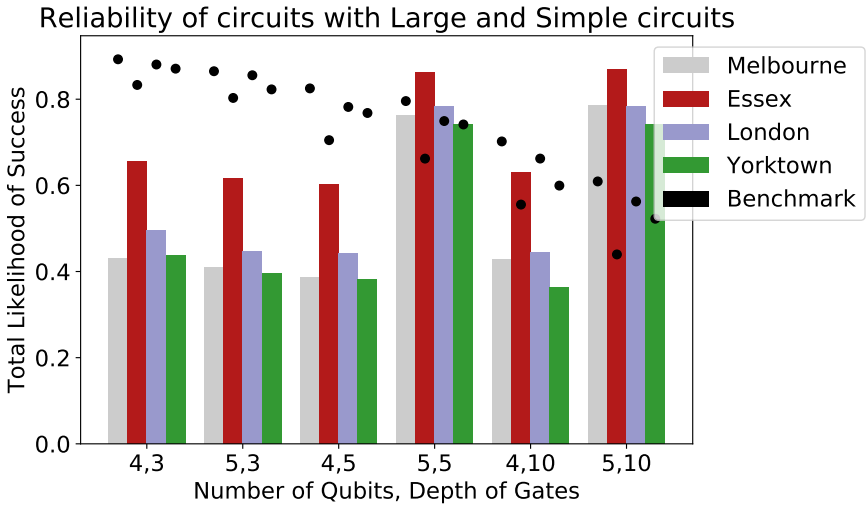


Figure 6.5: *A bar graph showing the likelihood of success of a single run of each generated circuit for large amounts of qubits (4-5) and simple gate complexity (max gate depth of 3-10) on 4 types of hardware.*

6.3.3 Benchmarking Examples

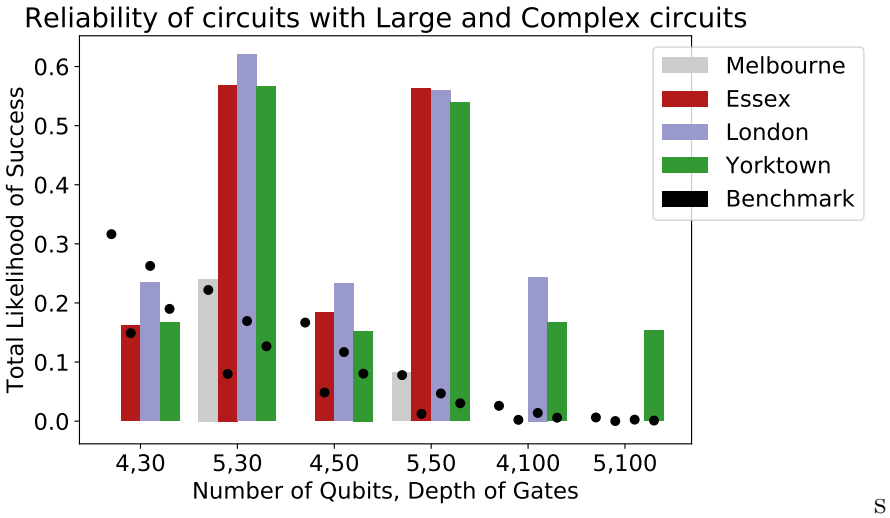
This section starts by looking at the examples of qasm code written for specific hardware. These have a variety of different complexities, varying from completely random sets of gates, to simple circuits such as swap or teleportation, to fully implemented algorithms such as Grover's algorithm. The idea behind this is that these algorithms are built specifically for a single hardware, we aim to see how qiskit's transpiler compares to the introduced PDDL schedule approach.

6.3.4 Overall Results

We start by looking at the overall performance of the PDDL compilations in comparison to the standard QISKit compilations. We also show a more focused view of the lower error circuits, to allow a better comparison by eye.

It is worth highlighting a few points here. In general, the PDDL compilations result in a higher likelihood of success than the standard qiskit compilations. On the hardware that the algorithm was designed for, the standard qiskit compiler performs very well, outperforming the PDDL in some cases. However, the tradeoff here is that for the rest of the hardware backends the PDDL scheduler outperformed significantly. This is as expected, as manual algorithm generation is expected to perform better than purely automated. Here the improvements offered in flexibility when transpilation show an advantage to the PDDL compilation method.

We highlight the Grover algorithm and the quantum plaquette as two circuits



s

Figure 6.6: *A bar graph showing the likelihood of success of a single run of each generated circuit for large amounts of qubits (4-5) and simple gate complexity (max gate depth of 30-100) on 4 types of hardware.*

where the qiskit compilation introduced considerable error during synthesis whereas the PDDL method did not.

For many very simple algorithms, both methods achieve identical minimal solutions.

It is notable that the 5 Qubit Essex (5QEssex) machine performs quite poorly with the PDDL scheduling. The 5QEssex has a low difference in quality between qubits as well as an inflexible structure of connections between them. This means that the advantages of using a method that aims to optimize around hardware are minimized.

6.3.5 Grover's Algorithm

Here we look at an example schedule of Grover's Algorithm on each of the three five-qubit backends. Each schedule is represented as a graph, with a block corresponding to any area where a gate is currently being run. The aim is to get a view of the density of the circuit and the differences the different hardware compilations result in. We start with the 5QYorktown, as it was the backend that this particular circuit was designed for.

The Yorktown machine has similar error rates across its qubits. Therefore, little is done to move around qubits. Instead, the focus is the maximum density of the gates to keep the run-time as short as possible. Notably, this results in identical error rates to the manual compilation.

We then look at the 5QLondon. Here, the circuit looks similar, with a single qubit

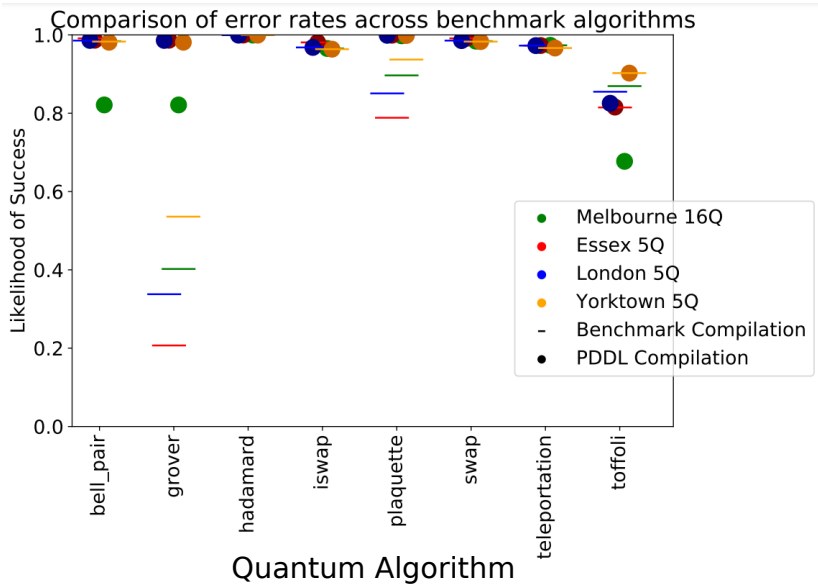


Figure 6.7: *A total view of a scatter graph showing the comparative performance of the PDDL method (circles) and benchmark method (lines) on various hardware. A zoomed-in graph is shown further below.*

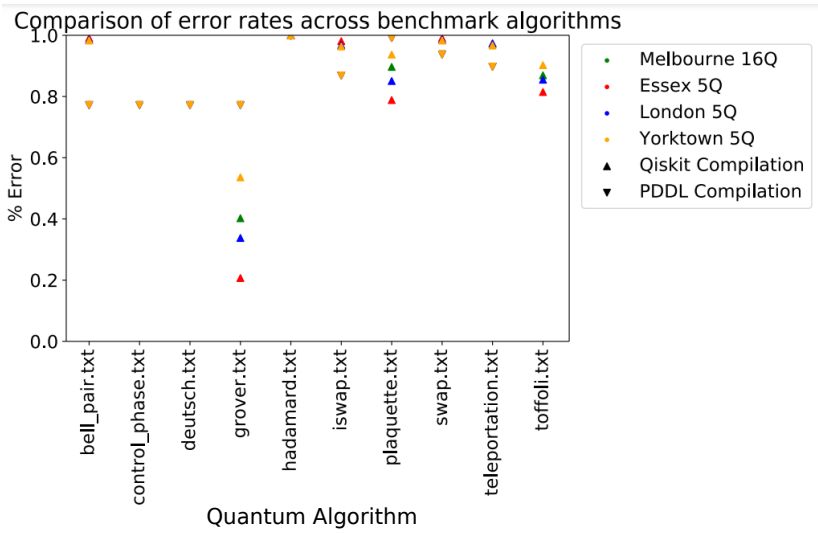


Figure 6.8: *A zoomed-in view of a scatter graph showing the comparative performance of the PDDL method (circles) and benchmark method (lines) on various hardware. This focuses on the top performing results.*

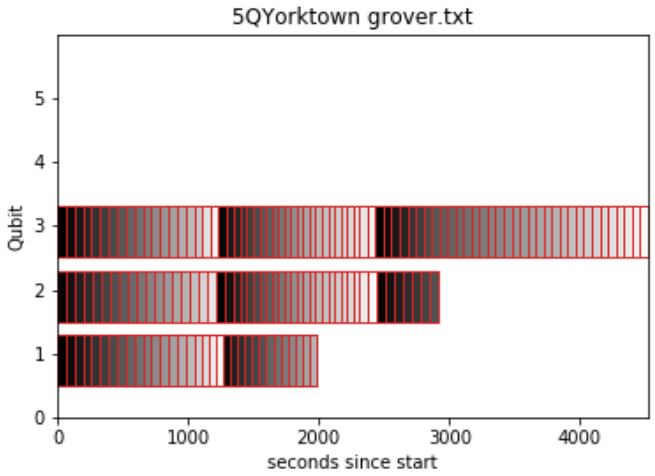


Figure 6.9: *A graph displaying the schedule of gates used to build Grover's algorithm on the Yorktown machine*

performing a considerable amount of the computation. However, it is notable that for this backend the second qubit was used rather than the third. This qubit performs computations with a lower average error rate, resulting in a better overall computation.

It is also notable that few early computations that were unordered were moved to be performed on qubits 4 and 5, which are unused in the original algorithm as well as the other circuits. The result is a circuit that performs significantly better than its qiskit compiled equivalent.

Finally, we look at the 5QEssex. This circuit results in a similar pattern to the 5QLondon. However, two things are notable. First, no calculations are done on qubits 5 and 4. In addition, there are visible gaps in qubits 3 and 1 where no computations are performed. 5QEssex's architecture makes swapping between the non-central qubit very difficult which limits flexibility. This generally lowers the quality of results found on it with non-bespoke methods. Since our scheduler aims to be as generalized as possible, this made it a poorer fit than normal.

Notably, several of these consist of schedules for what can be considered singular gates. This produces sets of structured rewrites of abstract gates in terms of hardware instructions. In other words, these output a rewrite rule for more complicated gates. When discussing the PDDL software stack, it is notable that the rewrite rules were not automated yet. While these features are out of the scope of this paper, it is noteworthy that the structure was designed with this in mind.

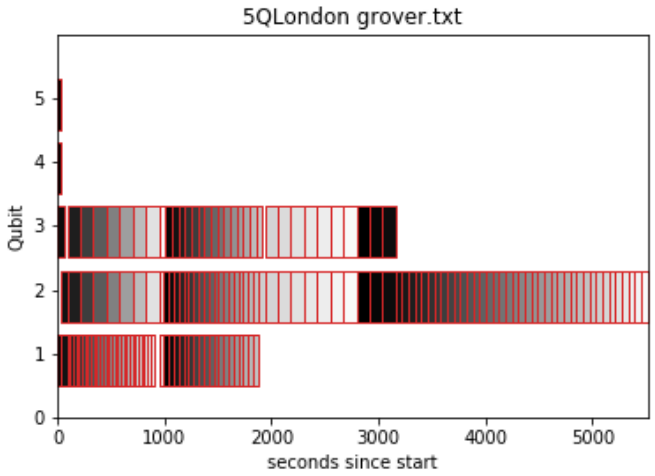


Figure 6.10: *A graph displaying the schedule of gates used to build Grover's algorithm on the London machine*

6.4 Chapter Summary and Final Thoughts

We demonstrated a method for automating the scheduling of gates and qubits when developing quantum programs. We took a combination of hardware and software specifications and automatically parsed them to translate them into PDDL models. These models were then automatically scheduled on qubits. The automatic scheduling method takes into consideration hardware quality and aims to minimise time and error. We tested this on a set of programs and with the specifications of real quantum machines.

The result of this is a fully automated process from hardware specification to circuit layout. In addition, the features of this fully automated process are modular. The quantum code and backend are fed into a parser, which produces a PDDL problem. This matches with a PDDL problem file to make a complete PDDL model. This PDDL model can then run on any PDDL planner. This automated, modular process is much easier during development. As working on each area requires much less widespread expertise. In addition, further developments in planning domains are more easily integrated.

In addition, while we expected this method to perform worse than hand-tuned and optimized methods, on larger problems it performed better. Automated reasoning of heuristics can be applied faster than hand-tuned methods or stricter optimization. By automating the design process, it allows larger problems to take advantage of the targeted design. This in turn results in better structured, more efficient problems.

This was all done with what ends up being a relatively simple method of automated reasoning. The objectives of this work favoured generalization over performance. The

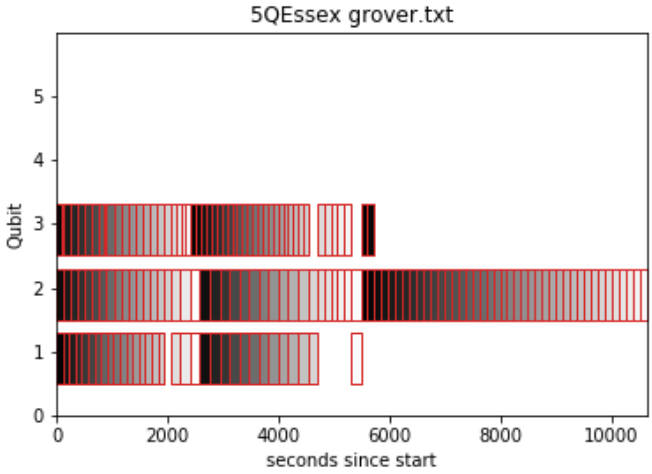


Figure 6.11: *A graph displaying the schedule of gates used to build Grover's algorithm on the Essex machine*

foundations are now provided for more advanced methods that could further strengthen this approach. This could include advanced rewriting logics, better measures of circuit assessment or search methods that better minimize the search space. Of particular note is that only minimal rewriting was done of gates. Generating rewrite rules is still difficult and still performed better as hand-calculations. However, for less general problems the same structure of approach here can be improved by hand-developing problem-specific rewrite rules which can provide much more flexibility to the system.

Chapter 7

Automated Software Stack Analysis

7.1 Chapter Introduction

In our third chapter, we look at a method of automating software engineering decisions when synthesising a quantum circuit. We do this by taking a step of the current development process that requires human construction and presenting an automated alternative. Specifically, we look at building a circuit transpiler stack using a classifier program.

We identify earlier in this thesis the need for stronger software engineering tools in quantum software development. In particular, we highlight the need for these tools to simplify areas that currently require human insight. Selecting an optimizer requires an insight into the structure of the quantum circuit being optimized. Our classification method allows a user without this insight to circumvent the requirement while still benefiting from circuit optimization.

In this chapter, we look at automating the construction of quantum circuit optimization pipelines. These methods are currently done by hand or by selecting from a small set of default options. We look at improving this by using automatic classification to build a bespoke stack to target a specific quantum circuit based on surface properties. We implement this using a support vector machine-based classifier. This is fully implemented in a way compatible with SCIKit, allowing different classification kernels to be substituted.

We split this into two problems. The first is a series of multi-class classifications that require the selection of the best method from a set of prewritten lists. The second is a multi-label classification that requires the selection of all methods that would result in some improvement.

Based on our results, it is clear that generating transpiler stacks with automatic

classifiers is a promising approach to simplifying their generation. We measured the odds of this method being able to select the best method from preselected lists. We found that the system predicted better than our benchmark odds. In addition, the multi-label portion of the classification performed very well with results varying between 80% and 60%. The success of both kinds of selection provides evidence of the viability of more complicated methods of selection (e.g. best three). As with the previous chapters, we highlight that one of the major advantages is that this approach does not require any user knowledge of how the transpiler methods work or the structure of the quantum circuit.

We expect this work to improve the speed and ease at which transpiler stacks can be implemented. In addition, by reducing the requirement to understand the methods involved in transpilation, the expertise barrier to entry for quantum circuit implementation is lowered. Finally, our method is implemented in a way that is compatible with standard SCI-Kit classification structures. This allows different classification methods to be applied with much greater ease. We hope that our work encourages further interest from machine learning communities to build increasingly sophisticated methods to approach this problem.

7.1.1 Software Stack Analysis

In chapter 3 we discuss program synthesis and in section 3.2 we look at how a software stack can be used in quantum program synthesis. In this chapter we are discussing analysis of quantum software stacks themselves. We therefore use the QISKit language as our example quantum software stack. The stack starts with a high-level descriptor of behaviour. This is then converted to a matrix representation of the program. This representation is rebuilt with ordered quantum gates. These quantum gates are then rewritten to target a specific quantum hardware in a process called transpilation. This now leaves the program as a series of timed instructions for physical actions that a real machine can take.

Building a Transpiler

The transpilation step is much more complex than previously mentioned. The main goal of the transpilation step is to rewrite the theoretical program to one that can now be run on a real machine. However, it also provides a secondary objective as the most sensible place to optimize the code in preparation for run-time. This means that it is necessary to treat the transpilation step as an optimization problem.

Transpilation can be built by selecting an ordered list of pre-written methods from an available pool. QISKit has a transpilation pipeline manager and the requisite pool of methods. These methods can be categorized as one of the following: layout initialization, gate to qubit routing, or post-optimization methods. Each pass requires

exactly one layout initialization and routing method. Any amount of optimization methods may also be added to the pass.

What Makes a Good Pipeline?

If we consider transpilation to be an optimization problem it is necessary to discuss what we are optimizing. There are two areas to look at here. First, we need to consider what we are optimizing within the circuit. This can range from various types of error, circuit size or number of gates among other things. Secondly, we need to consider optimizing the pipeline itself. The primary consideration for this is keeping the time to run the pipeline to within a reasonable bound. This also includes scaling considerations of the pipeline.

7.1.2 Classification AI

The aim of a classification AI is to take a set of data point structures and identify which of a set of categories they belong to. Datapoint structures can be built in a variety of ways but commonly focus on taking a number of measurable features from a larger system. For example, it may be difficult to compute with real humans so we might only look at their height and weight. Feature selection must strike a balance between accurately representing its source while also being small enough to perform large-scale computations with. Classifications are much simpler, as almost anything can be used interchangeably. The main distinction is whether the classifications are externally decided or automatically generated after looking at the dataset.

Classifiers have become a standard in computer science fields, with a range of tools available to implement a multitude of methods. One of the biggest challenges this provides is deciding which approach is most appropriate for the dataset. This will be discussed further alongside our model description, where it is easier to draw direct relations. However, it is still useful to highlight a few considerations here to guide our initial decisions. What is the size of the training data and how easily can it be wokred through? How objective an assessment of the result is possible? How big a concern is the time to run or train the model? To what extent does exploitable linearity appear in the dataset. How many features are in the data and how much would be lost by reducing the feature count?

7.1.3 Multiclass and Multilabel Classifications

Multiclass classifications are a simple extension of the standard classification problem where the aim is to choose which of a set of categories is the most appropriate. Multi-label classifications are a subtle further extension where each classification label is independent. This means that the problem is to take a list of labels and decide which, if any, are applicable for the current piece of data. This means that for a multiclass

problem, exactly one label must be chosen from the list of possible classifications. For a multi-label classification, any amount of appropriate labels may be chosen.

More formally the process of multi-label classification is the problem of creating a model to map inputs (x) to matching binary vectors (y). Each value of 1 in the binary vector represents that the label in the same position is applicable, whereas a value of 0 means the label is not.

One of the practical implications of this is that the result of a multilabel classification is more complicated to assess than a multiclass or single class problem. In single or multiclass problems, each classification is either correct or incorrect. In multilabel, there can be a spectrum of correctness. If a classifier gets two out of three labels, that is a better result than only finding one or zero.

7.2 The Data Model

7.2.1 Selecting Features from a Quantum Circuit

The main limitations of feature selection for a quantum circuit are the relatively low amount of available data as well as the inherent complexity of a quantum circuit representation. The main features of a quantum circuit are the quantum gates that compose it. While these contain a great deal of information (position, value, connections, etc.), the simplest implementation is simply a count of each type of quantum gate. This provides the additional advantage of maximising the available data: any quantum circuit can be used if it has been defined without any requisite benchmarking information. It is important to remember that the aim is a method that minimizes required expertise, so minimizing the input complexity is a necessary step.

7.2.2 The Optimization Pipeline

The optimization pipeline requires a restricted structure built from a combination of pre-written methods. Firstly, an initial mapping of theoretical qubits to real qubits must be done. This is called layout initialization. There are three available layout classifications. Then swap gates must be added in to move gates across to qubits where they can be physically run. This step is called routing. There are three available routing options. Finally, any amount of optimization methods can be run (including zero). There are eight available optimizations. These classification options are chosen from the transpiler passes available on Qiskit [87]. This gives a total search space of $4 \times 3 \times 2^8$ or 3072 possible options per circuit.

The layout methods were TrivialLayout, DenseLayout, SabreLayout (set to max iterations if 4) and CSPLayout. Ties were decided with in this order too, with Trivial-

Layout taking precedence over Dense and so on. The tie-breaking order was decided by ordering methods based on their computation time. That is to say we prefer methods with low-computing times. We also note that TrivialLayout performs a naive approach, this is added in to provide the option of doing nothing to systems that aren't predicted to benefit from optimization.

The Routing methods were BasicSwap, StochasticSwap (set to four trials) and LookAheadSwap (set to a width and depth of five). This is once again the order of tie-breaking. In addition, we have the equivalent BasicSwap to look for circuits that don't benefit from advanced routing. Stochastic swap looks at random search for simple optimizations and Lookahead swap aims to optimize circuits where some planning is required to build an efficient circuit.

The Optimization methods were Optimize1qGates, Collect2qBlocks, ConsolidateBlocks, CXCancellation, CommutationAnalysis, CommutativeCancellation, RemoveDiagonalGatesBeforeMeasure, RemoveResetInZeroState. This provides a mix of different approaches to simplifying the circuits. Some of them are very trivial (removing gates that aren't doing anything, while some are more complicated (analysing circuits for commutation).

7.2.3 Representing as Labels

The routing and layouts must have exactly one solution and therefore fall into a standard multiclass problem. The optimization is, however, a multilabel problem as any amount of labels can be applied. It is therefore worth looking at the problem as two distinct problems. The first applies a multiclass approach to select the predicted most effective routing and labelling approach. In addition, a prediction is made for the best optimization, aiming to provide exactly one value for the most effective method. We then look at applying a multilabel approach to just the optimizations category, trying to select any method which would improve the circuit.

7.3 Description of Method

For our initial multiclass problem, we used a one vs all classifier using python scikit. In order to implement this, the classes were converted into a unique binary code. This results in a single output containing only the class with the best fitting prediction. This provides a method where it is simple to swap out the classification approach for any binary classification method, while still providing a good benchmark for the expected performance on a dataset with unknown properties.

Our multilabel problem used a multioutput classifier to perform a single classification per label. For each circuit, a binary array is created with one value per label. Each

1 indicates the label applies, while a 0 applies that it does not. Applying a classifier to each individual value provides a simple method of extending multilabel functionality to otherwise incompatible algorithms.

To perform the classifications we used a Support Vector Machine (SVM). We highlight that the exact method of classification is not highly relevant to the work. An SVM was chosen for simplicity and compatibility reasons. It's straightforward to build and apply an SVM to multi-label classification problems and the expected results are likely to provide sufficient insight into the effectiveness of predictive methods.

7.3.1 Test Setup

Our tests look at a variety of different sized quantum circuits. We separate our tests to try and focus on how our approach performs on a variety of sizes. Smaller circuits result in smaller sets of features, meaning that classifiers will have less data to work with. Meanwhile, larger circuits may strain the search capacity of standard optimization techniques resulting in less diverse labelling as better scaling optimizations outclass other methods.

To differentiate the size of circuits we look at two features, the qubits and the gate depth. Increasing qubits increases the optimization search space more than it increases the number of features. Meanwhile, increasing the gate depth increases the size of the features vector more than it increases the optimization search space. We refer to qubit number as circuit size and refer to gate depth as circuit complexity.

Circuits were generated randomly including one- and two-qubit gates and with both measure and reset commands present within the circuit. To evaluate the correctness of a generated set of passes we used search to check all possible options. In order to shrink the search space the best routing and layout passes were selected first. After this step, the optimization passes were chosen using the routing and layout as a baseline. Each pipeline consists of exactly one layout assignment function, exactly one routing function and any amount of optimization functions. The optimization function that made the single largest difference in error when applied by itself was also highlighted.

7.3.2 Results

Low Gate Depth

We start looking at low-size, low-complexity circuits.

While the layout performs well enough at this small scale, this is actually much worse than when run on the larger systems. With small numbers of features, there is also only a small amount of data to differentiate the methods. With that in mind,

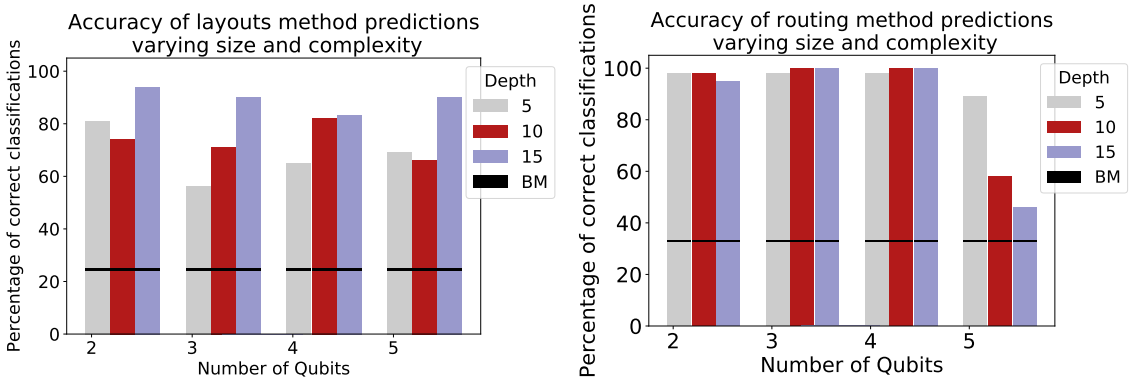


Figure 7.1: *The percentage of predictions of an SVM classifier that matched search-optimized solutions when selecting a layout method.*

the performance is still strong with **most sets having above a 60% predictive accuracy**. Even in the worst case, the best method is selected 50% of the time.

The routing method predictions return an above 90% accuracy between two and four qubits. At Five qubits, this performance drops significantly. In simple circuits, the routing method provides only very small improvements. Below four qubits, a basic routing methods almost always result in identical circuits. Since the tie-breaker is deterministic, this means that most of the values were looking at similar methods. At five qubits, we see the threshold where more advanced swaps can generate an advantage. However, at this borderline, the performance drops significantly.

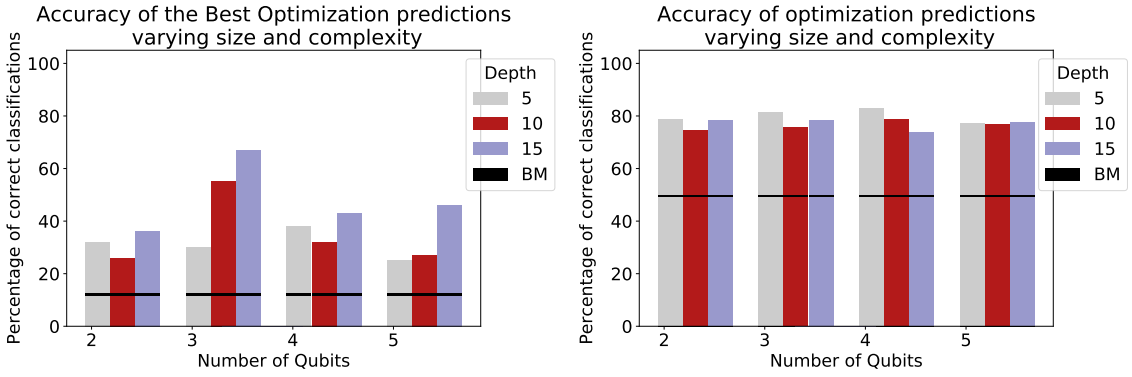


Figure 7.2: *The percentage of predictions of an SVM classifier that matched search-optimized solutions when selecting an optimization method for.*

Selecting the best optimization is meant to be the most difficult step. There are more optimization options than either of the previous two categories (8) and there is more significant overlap with their functionality. Very similar circuits may benefit

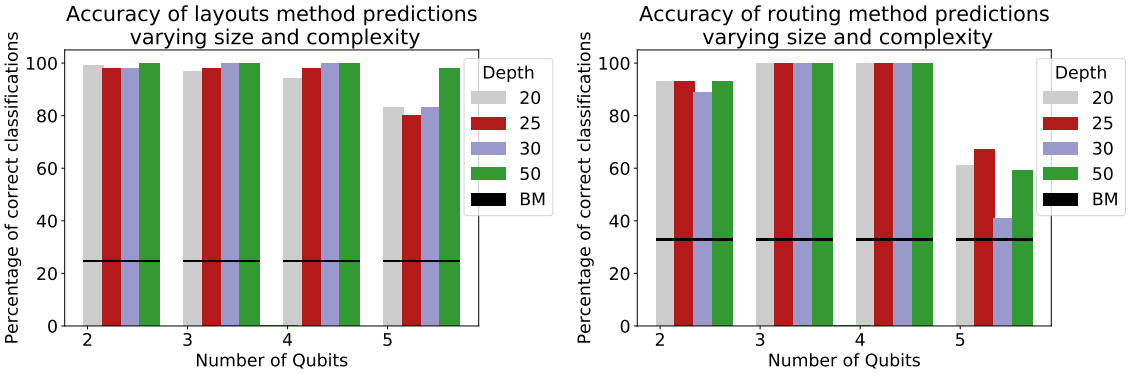


Figure 7.3: *The percentage of predictions of an SVM classifier that matched search-optimized solutions when selecting a layout method for large circuits.*

from one optimization but not another. However, overall, performance was still strong. Small qubit numbers appeared to make the most difficult classifications. However, the average performance was still strong with each result being higher than random chance by a statistically significant margin. In addition, certain circuit specifications had very high performances. This does indicate that **the regression method has generated a model that provides a useful insight to the system.**

For our final set of tests on smaller circuits, we look at identifying which of the 8 available optimizations will improve the system. As a reminder, this classification is a multi-label classification. This means that each optimization approach is either applied or not. The accuracy is measured as a percentage of how many labels are correct, regardless of whether it is a positive or negative label. Across the board, these returned a 60% accuracy for each prediction.

7.3.3 High Gate Depth

For the larger circuits with more features, the results are much stronger when predicting a routing method. **The classifier provides an incredibly high prediction accuracy of over 80% on all experiments.** We expected this to be one of the easier areas to classify, however it is still useful to see that the maximum potential is high.

The routing circuits follow a similar pattern on large circuits as it does on smaller ones. As before, the simplicity of a circuit with a small number of qubits (and qubit connections) means that routing methods have only limited room for improvement. This once again changes at 5 qubits, where the quality of predictions drops.

We then look at the optimization selection performs much better on larger circuits.

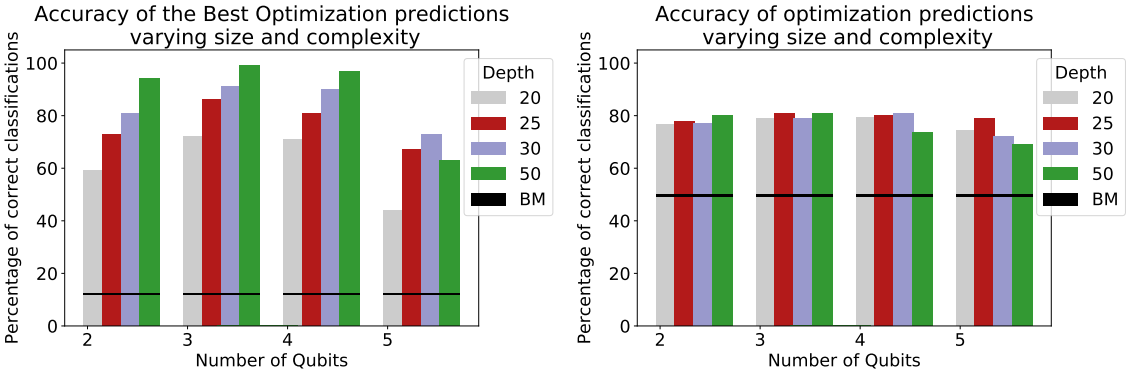


Figure 7.4: *The percentage of predictions of an SVM classifier that matched search-optimized solutions when selecting a routing method for large circuits.*

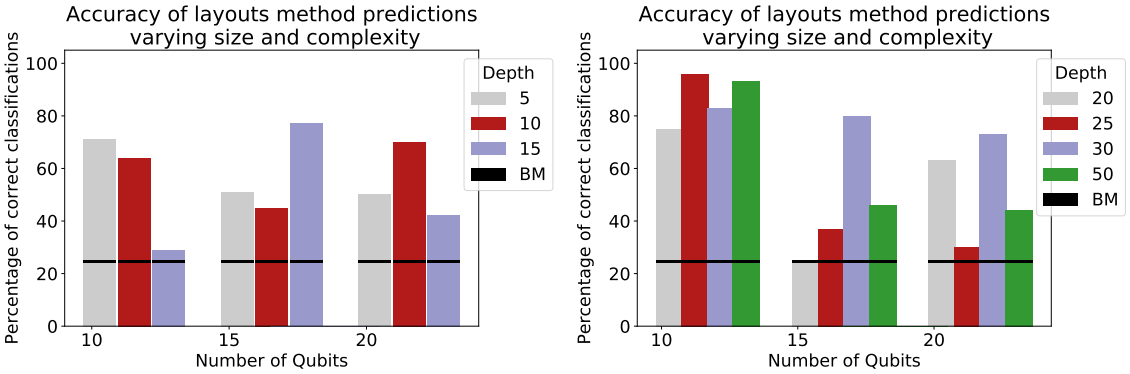


Figure 7.5: *The percentage of predictions of an SVM classifier that matched search-optimized solutions when selecting an layout method for circuits with high qubit counts.*

The increase in features outweighs the increase in complexity when the number of classes is so high. There is a clear trend that as the gate depth increases, the accuracy of the predictions also increases. In addition, we highlight the high quality of the predictions, with the majority of the results in the 60-90% range.

Finally, we look at the optimization selectors. We expected these methods to increase in performance compared to the smaller circuits. This is exactly how the results turned out, with the majority of values being almost 80% correct. Only a few results were lower, with the lower bound being 65% (i.e. still a strong result).

High Qubit Count, Low Gate Depth

At larger qubit counts, these systems become exponentially more difficult. As the absolute number of gates is a function of both the gate depth and the qubit size, the

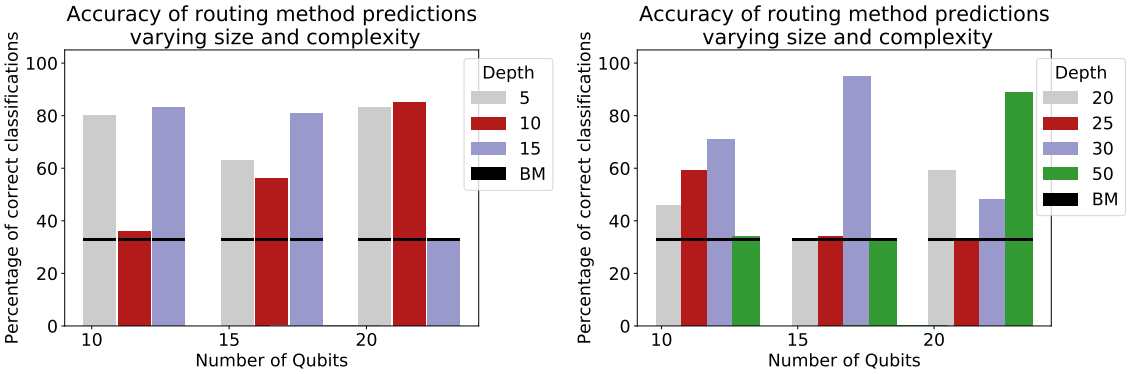


Figure 7.6: *The percentage of predictions of an SVM classifier that matched search-optimized solutions when selecting an optimization method for large circuits.*

absolute number of gates will increase. However, the prediction does not have location information on these gates. This makes it difficult to identify layout properties. Other than a single result, these were still statistically improved decisions.

The routing predictions have a similar result to the layouts. However, the main difference is a larger proportion of experiments resulting in a high degree of accuracy. It is unclear exactly what caused the two poor results in this set. However, this pattern was repeated with further tests on the lower values indicating that this is a pattern in the hardware. This is itself an interesting result. Neither of these profiles would indicate to a normal user that they would be difficult to classify, however, there is a consistently measurable difficulty.

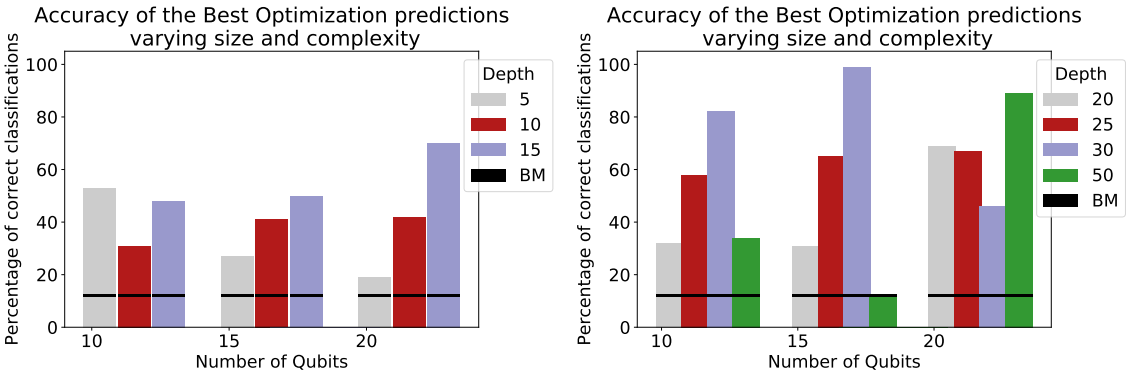


Figure 7.7: *The percentage of predictions of an SVM classifier that matched search-optimized solutions when selecting an optimization method for large circuits.*

Selecting a single best optimization is expected to be a difficult task but is also one

of the more important use cases of our methodology. Of these results, the 20 qubit results are the most relevant. This represents a reasonably sized set of quantum gates being run on the entirety of a quantum machine. And here we see strong results, the best of eight methods is successfully identified almost 50% of the time in the worst case and over 80% in the best case. This is further notable as the best case represents the largest sets of gates. Most results have shown that more information on features produces better results, indicating that with a larger data set there is a lot of potential for a scaling prediction engine.

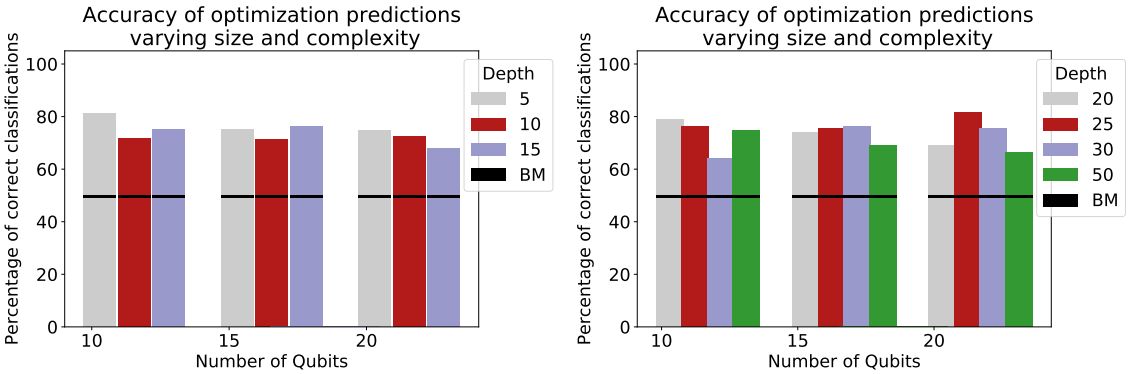


Figure 7.8: *The percentage of predictions of an SVM classifier that matched search-optimized solutions when selecting an optimization method for large circuits.*

Our final set of results returns to a multilabel classification problem. These results perform similar to the smaller circuits, with all results within the 60% to 80% range. Overall, this is a promising result. This functionality allows the automatic selection of methods from a large set in a single step, whereas the previous methods can only look at a single function. This means that these results being strong indicates a use case with applicability to much larger stack-building problems.

7.3.4 Results Summary

Overall, we see sets of results with a strong ability to predict the best selection for each method. When the circuit size increases, these predictions maintain their accuracy. Four out of forty-nine of the experiments failed to achieve any kind of insight into the system.

7.4 Chapter Summary

We have demonstrated a classification method that automatically builds a software pipeline for quantum transpilation. We have taken this method and applied it to various sized quantum circuits. This application aimed to choose methods that would

minimize the total error of the circuits. Our results showed that these predictions were overall accurate and provided high-quality pipelines for common sizes of modern quantum programs.

In addition, we highlight that these optimization steps can be run by non-experts. This is useful as it provides better default options to non-experts that would result in a lower error system. In addition, this can also provide a stronger starting point for expert-led optimization. Not only does the initial structure give a better prediction of what the final stack might look like, but it is also possible to draw insight into the circuit structure by looking only at the classifier output. Both of these lower the time and complexity of optimizing quantum circuits.

Finally, this model was integrated with a scikit SVM structure. This includes a method of converting standard QISKit hardware models into a compatible format for generic binary classifiers. This allows further work in improving the domain-specific classification by either experimenting with further classifier methods or through the specialization of the SVM kernel. Either way, this should help to make quantum optimization more accessible for classification research.

In particular, this structure is compatible with a multi-layer perceptron model of classification. This enables the implementation of neural networks and, on a larger scale, deep learning methods. These methods naturally fit the information that is encodable for quantum circuits. That is to say, it is straightforward for a human user to encode the structure of a quantum circuit and an objective assessment of quality while it is very difficult for a human to provide structure-related insight. Deep learning approaches provide powerful data-led decision-making abilities at the cost of not providing clarity of how they are making these decisions. This is useful in a field that already lacks clarity in the decisions while also benefiting greatly from the decision-making.

Chapter 8

Conclusions

8.1 Summary of Findings

8.1.1 Chapter 5

Chapter looks at applying a genetic improvement method to the problem of quantum circuit synthesis. Our approach started by parsing QISKit implementations of quantum circuits. These are then translated into ZX-graph representations. This representation allows for generic rewrite rules to be applied to the graph. These rewrite rules result in a new circuit with potentially different properties. By applying rewrite rules as mutation functions to a population of ZX-graphs, this can be done to efficiently filter through the search space. This was implemented in python.

This method resulted in an overall reduction in error compared to a standard quantum compilation. The results are particularly interesting for two reasons. First, the method can be automatically applied to any standard description of IBM-Q hardware and an input circuit description. As part of the experiments, a set of real quantum algorithms were transpiled to new machines with great success.

This method only requires a standard circuit and hardware model input, both of which are accessible and is therefore both a simple and effective method to implementing quantum code. Secondly, genetic improvement has had a long theoretical justification for application in quantum computing. It is therefore very useful to provide a proof of concept.

In addition to the usefulness of the method is also provides a useful case study on the advantages of ZX. ZX is a controversial language in the wider research community. While it has a strong theoretical backing there it has been historically difficult to find practical work using it. While this has been changing more recently [41], there is still plenty of discussions to be had about whether ZX is a promising method of representing and manipulating quantum circuits. While ZX is not a perfect representation, the

features provided by a graph-based representation with flexible rewrite rules provided necessary properties. The research performed provides a practical example to highlight the value of graph-based representation.

8.1.2 Chapter 6

Chapter 6 looks at building a planning model to solve quantum scheduling problems. The key result is the generated schedules that provide instructions for a quantum circuit. Once again the **results shows an overall reduction in error**. In addition, the results provide an interesting structure that integrates the quantum circuit problem into a planning domain model. This lowers the knowledge barrier to entry to allow non-quantum experts to target quantum problems with planning AI technology.

8.1.3 Chapter 7

Chapter 7 looks at building an automated classifier to suggest the best optimization methods to build a software stack. The results generated show the classifiers' ability to predict whether a method would be effective. The **results show the methods high accuracy which was as high as 60% to 80%**. This method is interesting as it does not aim to replace hand-optimization but work alongside it. This helps automate expert decision making by providing initial structures without requiring expertise. This helps simplify the design process by lowering the workload of experts and also extending the range of work that partial-experts can perform.

8.1.4 Discussion of Research Aims

Provide proof of concept of automated methods of building quantum software

In this work, we provided three proof of concept implementations of automated methods of building quantum software. The first method is automated circuit transpilation. The second method is automated qubit scheduling. The final method automates the software design process. In addition to the proof of concept of implementation, these also passively identify three areas of future focus for quantum software automation research.

Integrating these methods into practical software stacks that provide clearer, more flexible structure

The work of each of the three chapters has provided steps to integration with software stacks. The first method provides a first step into software structure by approaching quantum compilation as a multistage process, taking a single step and substituting a new approach in. The second method builds qubit scheduling directly into a PDDL model with surrounding parsing software to integrate quantum hardware and software.

This enables the assimilation of new planning AI technology as it is developed. The final method acts directly on a software stack and looks at directly building a software pipeline. As new methods are developed, this means they can be automatically prioritized and assigned.

Ensure that these methods are compatible with the real quantum systems, being able to both automatically integrate hardware and build off state-of-the-art software

This final research goal is accomplished by integrating each method with IBM-Q software and hardware. Each approach can automatically parse the hardware specifications of a machine and can output a QISKit specification that is physically implementable on the targeted backend.

8.2 Limitations

8.2.1 Size and Scalability of Results

The experiments run in all three of the contribution chapters are performed on relatively small machines. The circuit data generated is also comprised of generally small circuits. On these smaller circuits, the performance has been acceptably low. In addition, the three methods used are commonly used on large, scaling data sets. It is therefore expected that this would be maintained for the circuit synthesis problem. However, it is worth noting that no proof of scaling was generated for any of these methods. This means that theoretically, there is no guarantee that these methods will scale. While we did not consider this necessarily relevant, as we take an empirical perspective, it is still an important consideration to highlight.

8.2.2 Applicability to Future Quantum Computers

Modern NISQ computers are an exciting stage of quantum computing development. However, there are expectations that future machines will eventually outclass them. Improvements to error correction, noise-reduction and quantum hardware mean that future quantum machines may look unrecognizable compared to current hardware. There is a legitimate question of how applicable this thesis' findings are to different or new machines. Notably, we assume a gate based representation of the quantum circuit exists which further requires additional assumptions. For example, a gate based model presumes that any physical action of the hardware can be considered as an independent action applied to a finite set of qubits and that each action happens without overlap.

8.2.3 Availability of Quantum Circuit Data

There is a very limited set of quantum circuits that are practically available. While certain repositories do exist, these are mainly simple circuit structures or benchmarking circuits. At present, there is simply a lack of commercial quantum circuits to provide completely realistic data. This results in two limitations. First, for large batches of data we used generated quantum circuits. These may lack the natural structure of designed circuits, even if the surface properties are very similar. Secondly, when benchmark circuits were used, the sample size was much smaller than what would be optimal. Combined these give sufficient insight into the systems, but the results must still be taken with a grain of salt.

8.2.4 Measuring the Quality of Circuits

Each of the chapters focuses on a single measure of the quality of the generated circuits. This is the combined error rate introduced by the gates that build up the circuit. For the purpose of gate synthesis, this is a useful measure of quality. However, it is far from the only one. Other metrics, such as total-run time, the longest span of gates and the total number of gates all provide some insight into the quality of a quantum circuit.

While the total gate error was a useful metric to see the changes from reorganizing only the gates, it is but one facet of what quality might be understood to be. Future work should investigate the advantages of other quality metrics to assess new circuits.

8.3 Further Work

8.3.1 Chapter 5

ZX replacement

In chapter one, one of the more interesting elements of our approach was the use of the ZX-calculus. The use of graph-based representations greatly benefits the use of genetic algorithms. Rewrite rules can be applied without changing the core functionality of the circuit. This allows the genetic algorithm to apply mutations but maintain the assumption that the circuit is identical. This provides a method to circumvent the need to recheck the circuit equivalency at the evaluation step. This would otherwise be an incredibly expensive step of the computation.

This high degree of utility is what led us to use a ZX-graph representation. However, opinions of ZX in the wider quantum research field are mixed and rightly so. ZX is a highly abstract and generalizable representation but there are still problems in translating ZX graphs to practical circuits. In our approach, we constrained the properties of ZX graphs to mitigate this but it is still an imperfect solution. It is therefore

worth further research into developing a representation to use in this area instead of ZX. To do this, advantage can be taken of the fact that ZX is a very abstract language aimed at a very general case of quantum circuits. We can therefore look at replacing it with a specific use-driven representation built specifically for IBM gate-based circuits.

8.3.2 Chapter 6

Scaling PDDL Approach

In chapter 6, we generate a PDDL model of a qubit scheduling. This approach performed well providing accurate descriptions of hardware usage during the program run-time. These were successfully applied to what would currently be considered quite large quantum circuit problems. However, it is likely that future quantum systems are going to be much larger still. The search space gets much larger as the quantum system scales. The current model of a quantum circuit is abstract and does not convey all of the physical limitations. This means that some of the actions that are obviously not applicable to a human can not be immediately rejected by a human. These obviously impossible actions make up the majority of the search space. This presents an opportunity to shrink the search space significantly by proactively pruning infeasible actions.

One of the advantages of PDDL models is that new planning AI methods can be immediately applied to the problems. This means that a domain-specific method can be generated to target quantum circuit synthesis. Specifically, it would be useful to target a method that can shrink the search space of the problem. Of these, the simplest method is to prevent the consideration of groups of actions that are clearly impossible and integrating basic inference for what is likely to be possible. An example of this would be assuming that a quantum state will not move unless acted on. This is an obvious property to a human but not something that a generic AI can assume. This can greatly reduce the computational cost of running PDDL schedules.

8.3.3 Chapter 7

Deep Learning

In chapter 7, one of the major contributions was the development of an AI pipeline with an exchangeable classifier core. For our experiments, we selected a Support Vector Machine as the most appropriate classifier. This was in large part because of the limited amount of data of quantum circuits. An SVM is expected to provide strong results even with the limited available data. However, quantum circuits are complicated machines meaning that circuits with very similar, or even overlapping, properties may require different classifications. SVM classifiers use a dividing hyperplane to separate data points. This makes them inherently struggle with problems that are not linearly separable.

One of the most promising alternatives to this is a multi-layered neural network (i.e. a deep learning model). Deep learning has been used on similarly complex problems, including classical circuit analysis [28]. It is capable of solving problems with large amounts of complexity, interchangeable or highly similar components and highly ambiguous scenarios. The main limitation is the high amount of data required to train these models. With the state of quantum circuit development at the time of this thesis, this data would not have been attainable. However, when it is, this provides a clear area for future improvement.

8.3.4 Overall Further Work

The overall takeaway from the further work section is that quantum software generation should still benefit from the same approaches that work on classical software generation. There are too many of these to talk about, ranging from parameter selection on the GI approaches to formal methods to sophisticated AIs. One of the easiest ways to improve quantum methods is to simply identify areas of the generation process to which there are classically analogous approaches. Adapting these methods is much simpler than completely rebuilding quantum methods.

In order to facilitate this, we encourage work to further break down the quantum software stack and identify problems that do not require bespoke quantum solutions. We also encourage researchers from non-quantum backgrounds to look at methods being applied to quantum problems. A repeated statement when presenting the published works of this thesis was that the problem domain was interesting but it was difficult to approach it. We hope the chapters help demonstrate that quantum integration of classical automation approaches is not as complicated as it initially appears.

8.4 Final Remarks

We have presented three contributions to the development of the automation of quantum software synthesis. Each of these works provides a structured proof of concept of a methodology compatible with real quantum software development methods. These provide foundation examples to build on when automating other areas of the quantum software stack. In addition, we highlight the need for a well-planned quantum software stack to enable quantum development. Finally, these tools are built in such a way that non-quantum experts may more easily contribute from their field. We encourage readers to consider the value of this when building future quantum software approaches and take proactive action to ensure that future methodologies help build towards an iterative, modular software stack. We also encourage readers to look at current quantum software approaches and consider whether stages of this process can be automated, whether for personal ease or to minimize the complexity of the system.

Bibliography

[1] M. Ahsan, R. V. Meter, and J. Kim, "Designing a million-qubit quantum computer using a resource performance simulator," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 12, no. 4, pp. 1–25, 2015.

[2] T. Albash and D. A. Lidar, "Adiabatic quantum computation," *Reviews of Modern Physics*, vol. 90, no. 1, p. 015 002, 2018.

[3] P. Aliferis, "Level reduction and the quantum threshold theorem," *arXiv preprint quant-ph/0703230*, 2007.

[4] *Backend specification v1.1.0*, `https://github.com/Qiskit/ibmq-device-information`, 2018.

[5] *Backend specification v1.1.0*, `https://github.com/Qiskit/ibmq-device-information`, 2018.

[6] J. S. Bell, "On the einstein podolsky rosen paradox," *Physics Physique Fizika*, vol. 1, no. 3, p. 195, 1964.

[7] P. Benioff, "The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines," *Journal of statistical physics*, vol. 22, no. 5, pp. 563–591, 1980.

[8] ——, "The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines," *Journal of statistical physics*, vol. 22, no. 5, pp. 563–591, 1980.

[9] J. Benton, A. Coles, and A. Coles, "Temporal planning with preferences and time-dependent continuous costs," in *Twenty-Second International Conference on Automated Planning and Scheduling*, 2012.

[10] Z. Bian, F. Chudak, R. B. Israel, B. Lackey, W. G. Macready, and A. Roy, "Mapping constrained optimization problems to quantum annealing with application to fault diagnosis," *Frontiers in ICT*, p. 14, 2016.

[11] L. S. Bishop, "Qasm 2.0: A quantum circuit intermediate representation," in *APS Meeting Abstracts*, 2017.

[12] K. Booth, M. Do, J. Beck, E. Rieffel, D. Venturelli, and J. Frank, "Comparing and integrating constraint programming and temporal planning for quantum circuit compilation," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 28, 2018.

[13] G. K. Brennen and A. Miyake, "Measurement-based quantum computer in the gapped ground state of a two-body hamiltonian," *Physical review letters*, vol. 101, no. 1, p. 010 502, 2008.

[14] H. J. Briegel, D. E. Browne, W. Dür, R. Raussendorf, and M. Van den Nest, "Measurement-based quantum computation," *Nature Physics*, vol. 5, no. 1, p. 19, 2009.

[15] ——, "Measurement-based quantum computation," *Nature Physics*, vol. 5, no. 1, pp. 19–26, 2009.

[16] Z. Cai, "Quantum error mitigation using symmetry expansion," *Quantum*, vol. 5, p. 548, 2021.

[17] E. T. Campbell, B. M. Terhal, and C. Vuillot, "Roads towards fault-tolerant universal quantum computation," *Nature*, vol. 549, no. 7671, pp. 172–179, 2017.

[18] S. Chand, H. K. Singh, T. Ray, and M. Ryan, "Rollout based heuristics for the quantum circuit compilation problem," in *2019 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2019, pp. 974–981.

[19] A. M. Childs, R. Kothari, and R. D. Somma, "Quantum algorithm for systems of linear equations with exponentially improved dependence on precision," *SIAM Journal on Computing*, vol. 46, no. 6, pp. 1920–1950, 2017.

[20] J. I. Cirac and P. Zoller, "Goals and opportunities in quantum simulation," *Nature Physics*, vol. 8, no. 4, p. 264, 2012.

[21] B. Coecke and R. Duncan, "Interacting quantum observables: Categorical algebra and diagrammatics," *New Journal of Physics*, vol. 13, no. 4, p. 043 016, 2011.

[22] Coecke and Duncan, "Interacting quantum observables," L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, Eds., pp. 298–310, 2008.

[23] I. Cohen and K. Mølmer, "Deterministic quantum network for distributed entanglement and quantum computation," *Physical Review A*, vol. 98, no. 3, p. 030 302, 2018.

[24] D. G. Cory, M. Price, W. Maas, *et al.*, "Experimental quantum error correction," *Physical Review Letters*, vol. 81, no. 10, p. 2152, 1998.

[25] A. Cross, "The ibm q experience and qiskit open-source quantum computing software," in *APS Meeting Abstracts*, 2018.

[26] A. Cross, A. Javadi-Abhari, T. Alexander, *et al.*, "Openqasm 3: A broader and deeper quantum assembly language," *ACM Transactions on Quantum Computing*, 2021.

[27] V. Danos, E. Kashefi, and P. Panangaden, "The measurement calculus," *Journal of the ACM (JACM)*, vol. 54, no. 2, 8–es, 2007.

[28] J. Dean, "1.1 the deep learning revolution and its implications for computer architecture and chip design," in *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*, IEEE, 2020, pp. 8–14.

[29] D. Deutsch, "Quantum theory, the church–turing principle and the universal quantum computer," *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, vol. 400, no. 1818, pp. 97–117, 1985.

[30] D. Deutsch and R. Jozsa, "Rapid solution of problems by quantum computation," *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, vol. 439, no. 1907, pp. 553–558, 1992.

[31] O. Di Matteo and M. Mosca, "Parallelizing quantum circuit synthesis," *Quantum Science and Technology*, vol. 1, no. 1, p. 015 003, 2016.

[32] V. Dunjko and H. J. Briegel, "Machine learning & artificial intelligence in the quantum domain: A review of recent progress," *Reports on Progress in Physics*, vol. 81, no. 7, p. 074 001, 2018.

[33] S. Endo, S. C. Benjamin, and Y. Li, "Practical quantum error mitigation for near-future applications," *Physical Review X*, vol. 8, no. 3, p. 031 027, 2018.

[34] R. P. Feynman, "Simulating Physics with Computers," *International Journal of Theoretical Physics*, vol. 21, no. 6-7, pp. 467–488, Jun. 1982. DOI: 10.1007/BF02650179.

[35] M. Fingerhuth, T. Babej, and P. Wittek, "Open source software in quantum computing," *PloS one*, vol. 13, no. 12, e0208561, 2018.

[36] M. Fox and D. Long, "Pddl2. 1: An extension to pddl for expressing temporal planning domains," *Journal of artificial intelligence research*, vol. 20, pp. 61–124, 2003.

[37] P. R. Giri and V. E. Korepin, "A review on quantum search algorithms," *Quantum Information Processing*, vol. 16, no. 12, p. 315, 2017.

[38] D. Gottesman and I. L. Chuang, "Demonstrating the viability of universal quantum computation using teleportation and single-qubit operations," *Nature*, vol. 402, no. 6760, pp. 390–393, 1999.

[39] A. S. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger, and B. Valiron, *Quipper: A scalable quantum programming language*, ACM, 2013.

[40] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, 1996, pp. 212–219.

[41] M. Hanks, M. P. Estarellas, W. J. Munro, and K. Nemoto, "Effective compression of quantum braided circuits aided by zx-calculus," *Physical Review X*, vol. 10, no. 4, p. 041 030, 2020.

[42] M. Hayashi and M. Hajdušek, "Self-guaranteed measurement-based quantum computation," *Physical Review A*, vol. 97, no. 5, p. 052 308, 2018.

[43] A. JavadiAbhari, S. Patil, D. Kudrow, *et al.*, "Scaffcc: Scalable compilation and analysis of quantum programs," *Parallel Computing*, vol. 45, pp. 2–17, 2015.

[44] T. Jones and S. Benjamin, "Quantum compilation and circuit optimisation via energy dissipation," *arXiv: Quantum Physics*, 2018.

[45] S. P. Jordan, "Quantum computation beyond the circuit model," 2008.

[46] T. Karzig, C. Knapp, R. M. Lutchyn, *et al.*, "Scalable designs for quasiparticle-poisoning-protected topological quantum computation with majorana zero modes," *Physical Review B*, vol. 95, no. 23, p. 235 305, 2017.

[47] V. Kendon, A. Sebald, and S. Stepney, "Heterotic computing: Past, present and future," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 373, no. 2046, p. 20 140 225, 2015.

[48] V. Kendon, A. Sebald, S. Stepney, M. Bechmann, P. Hines, and R. C. Wagner, "Heterotic computing," in *International Conference on Unconventional Computation*, Springer, 2011, pp. 113–124.

[49] A. Kissinger and J. Wetering, "Pyzx: Large scale automated diagrammatic reasoning," *Electronic Proceedings in Theoretical Computer Science*, vol. 318, pp. 230–242, Apr. 2020. DOI: 10.4204/EPTCS.318.14.

[50] E. Knill, D. Leibfried, R. Reichle, *et al.*, "Randomized benchmarking of quantum gates," *Physical Review A*, vol. 77, no. 1, p. 012 307, 2008.

[51] P. Kruchten, R. L. Nord, and I. Ozkaya, "Technical debt: From metaphor to theory and practice," *IEEE Software*, vol. 29, no. 6, pp. 18–21, 2012. DOI: 10.1109/MS.2012.167.

[52] W. B. Langdon and M. Harman, "Genetically improved cuda c++ software," in *European Conference on Genetic Programming*, Springer, 2014, pp. 87–99.

[53] D. A. Lidar and T. A. Brun, *Quantum error correction*. Cambridge university press, 2013.

[54] N. M. Linke, D. Maslov, M. Roetteler, *et al.*, "Experimental comparison of two quantum computing architectures," *Proceedings of the National Academy of Sciences*, vol. 114, no. 13, pp. 3305–3310, 2017.

[55] Y. I. Manin, *Vychislimoe i nevychislimoe (computable and noncomputable), moscow: Sov*, 1980.

[56] C. T. Maravelias and C. Sung, "Integration of production planning and scheduling: Overview, challenges and opportunities," *Computers & Chemical Engineering*, vol. 33, no. 12, pp. 1919–1930, 2009.

[57] A. J. McCaskey, E. F. Dumitrescu, D. Liakh, M. Chen, W.-c. Feng, and T. S. Humble, "A language and hardware independent approach to quantum–classical computing," *SoftwareX*, vol. 7, pp. 245–254, 2018.

[58] D. C. McKay, C. J. Wood, S. Sheldon, J. M. Chow, and J. M. Gambetta, "Efficient z gates for quantum computing," *Physical Review A*, vol. 96, no. 2, p. 022 330, 2017.

[59] D. Melnikov, A. Mironov, S. Mironov, A. Morozov, and A. Morozov, "Towards topological quantum computer," *Nuclear Physics B*, vol. 926, pp. 491–508, 2018.

[60] P. S. Menon and M. Ritwik, "A comprehensive but not complicated survey on quantum computing," *IERI Procedia*, vol. 10, pp. 144–152, 2014.

[61] M. Möttönen, J. J. Vartiainen, V. Bergholm, and M. M. Salomaa, "Quantum circuits for general multiqubit gates," *Physical review letters*, vol. 93, no. 13, p. 130 502, 2004.

[62] M. A. Nielsen and I. L. Chuang, *Quantum computing and quantum information*, 2000.

[63] H. Nishimura and T. Yamakami, "An application of quantum finite automata to interactive proof systems," *Journal of Computer and System Sciences*, vol. 75, no. 4, pp. 255–269, 2009.

[64] G. O'Brien and J. A. Clark, "Using genetic improvement to retarget quantum software on differing hardware," in *2021 IEEE/ACM International Workshop on Genetic Improvement (GI)*, 2021, pp. 31–38. DOI: 10.1109/GI52543.2021.00015.

[65] R. W. Ogburn and J. Preskill, "Topological quantum computation," in *NASA International Conference on Quantum Computing and Quantum Communications*, Springer, 1998, pp. 341–356.

[66] *Openqasm example pages*, https://github.com/openqasm/openqasm/tree/main/examples, 2022.

[67] F. Pastawski and B. Yoshida, "Fault-tolerant logical gates in quantum error-correcting codes," *Physical Review A*, vol. 91, no. 1, p. 012 305, 2015.

[68] R. Portugal, R. A. Santos, T. D. Fernandes, and D. N. Gonçalves, "The staggered quantum walk model," *Quantum Information Processing*, vol. 15, no. 1, pp. 85–101, 2016.

[69] J. Preskill, "Quantum computing and the entanglement frontier - rapporteur talk at the 25th solvay conference," Mar. 2012.

[70] ——, "Quantum computing in the nisq era and beyond," *Quantum*, vol. 2, p. 79, 2018.

[71] L. M. Procopio, A. Moqanaki, M. Araújo, *et al.*, "Experimental superposition of orders of quantum gates," *Nature communications*, vol. 6, no. 1, pp. 1–6, 2015.

[72] A. Przysieżna, M. Horodecki, P. Horodecki, *et al.*, "Quantum metrology: Heisenberg limit with bound entanglement," *Physical Review A*, vol. 92, no. 6, p. 062 303, 2015.

[73] R. Raussendorf and H. J. Briegel, "A one-way quantum computer," *Phys. Rev. Lett.*, vol. 86, pp. 5188–5191, 22 May 2001. DOI: 10.1103/PhysRevLett.86.5188. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.86.5188.

[74] M. Sarvaghad-Moghaddam, P. Niemann, and R. Drechsler, "Multi-objective synthesis of quantum circuits using genetic programming," in *International Conference on Reversible Computation*, Springer, 2018, pp. 220–227.

[75] J. Shalf, "The future of computing beyond moore's law," *Philosophical Transactions of the Royal Society A*, vol. 378, no. 2166, p. 20 190 061, 2020.

[76] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proceedings 35th annual symposium on foundations of computer science*, Ieee, 1994, pp. 124–134.

[77] ——, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.

[78] M. Sisodia, A. Shukla, and A. Pathak, "Experimental realization of nondestructive discrimination of bell states using a five-qubit quantum computer," *Physics Letters A*, vol. 381, no. 46, pp. 3860–3874, 2017.

[79] L. Spector, *Automatic Quantum Computer Programming: a genetic programming approach*. Springer Science & Business Media, 2004, vol. 7.

[80] K. Srinivasan, S. Satyajit, B. K. Behera, and P. Panigrahi, "Efficient quantum algorithm for solving travelling salesman problem: An ibm quantum experience," *arXiv: Quantum Physics*, 2018.

[81] A. Steane, "Quantum computing," *Reports on Progress in Physics*, vol. 61, no. 2, p. 117, 1998.

[82] ——, "Quantum computing," *Reports on Progress in Physics*, vol. 61, no. 2, pp. 117–173, Feb. 1998. DOI: 10.1088/0034-4885/61/2/002. [Online]. Available: https://doi.org/10.1088/0034-4885/61/2/002.

[83] J. Stolze and D. Suter, *Quantum computing: a short course from theory to experiment*. John Wiley & Sons, 2008.

[84] M. Streif and M. Leib, "Comparison of qaoa with quantum and simulated annealing," *arXiv preprint arXiv:1901.01903*, 2019.

[85] F. Tacchino, C. Macchiavello, D. Gerace, and D. Bajoni, "An artificial neuron implemented on an actual quantum processor," *npj Quantum Information*, vol. 5, no. 1, pp. 1–8, 2019.

[86] Q. D. Team. "Transpiler." (), [Online]. Available: `https://qiskit.org/documentation/apidoc/transpiler.html`.

[87] *Transpiler passes*, `https://qiskit.org/documentation/apidoc/transpiler_passes.html`, 2018.

[88] D. Venturelli, M. Do, B. O'Gorman, *et al.*, "Quantum circuit compilation: An emerging application for automated reasoning," 2019.

[89] D. Venturelli, M. Do, E. Rieffel, and J. Frank, "Compiling quantum circuits to realistic hardware architectures using temporal planners," *Quantum Science and Technology*, vol. 3, no. 2, p. 025 004, 2018.

[90] D. Venturelli, M. Do, E. G. Rieffel, and J. Frank, "Temporal planning for compilation of quantum approximate optimization circuits.," in *IJCAI*, 2017, pp. 4440–4446.

[91] P. E. Vermaas, *The societal impact of the emerging quantum technologies: A renewed urgency to make quantum theory understandable*, 2017.

[92] J. Ward, J. Otterbach, G. Crooks, N. Rubin, and M. da Silva, "Qaoa performance benchmarks using max-cut," *APS*, vol. 2018, R15–007, 2018.

[93] D. Wecker and K. Svore, "Liquid: A software design architecture and domain-specific language for quantum computing," Feb. 2014.

[94] J. van de Wetering, "Zx-calculus for the working quantum computer scientist," *arXiv preprint arXiv:2012.13966*, 2020.

[95] N. Wirth, "Program development by stepwise refinement," in *Pioneers and Their Contributions to Software Engineering*, Springer, 2001, pp. 545–569.

[96] L. Wossnig, Z. Zhao, and A. Prakash, "Quantum linear system algorithm for dense matrices," *Physical review letters*, vol. 120, no. 5, p. 050 502, 2018.

[97] F. Yan, A. M. Iliyasu, Z.-T. Liu, A. S. Salama, F. Dong, and K. Hirota, "Bloch sphere-based representation for quantum emotion space," *Journal of Advanced Computational Intelligence and Intelligent Informatics*, vol. 19, no. 1, pp. 134–142, 2015.

[98] A. C.-C. Yao, "Quantum circuit complexity," in *Proceedings of 1993 IEEE 34th Annual Foundations of Computer Science*, IEEE, 1993, pp. 352–361.

[99] X.-C. Yao, T.-X. Wang, H.-Z. Chen, *et al.*, "Experimental demonstration of topological error correction," *Nature*, vol. 482, no. 7386, pp. 489–494, 2012.

[100] Y. Zhang and Q. Yuan, "A multiple bits error correction method based on cyclic redundancy check codes," in *2008 9th International Conference on Signal Processing*, IEEE, 2008, pp. 1808–1810.