ABSTRACT

WILSON, ELLIS JOHN. Software Techniques for Improving the Accessibility and Performance of NISQ Era Quantum Computing. (Under the direction of Frank Mueller).

Quantum programming is full of challenges on today's quantum hardware. Not only does one have to contend with small, error prone chips, but every type of quantum computer requires different skills to be learned before programming them.

In this work, we provide three improvements that can be performed on an individual problem basis using only open source software.

First, we demonstrate how measuring qubit noise immediately prior to running a small circuit can greatly reduce noise when compared to trusting the hardware provided noise values. This benefit comes from the process of transpilation, or the selection of which qubits to use to run the circuit. Selecting qubits which perform better can have a dramatic impact on the results of the circuit.

Second, we show the creation of "approximate circuits" —circuits which contain fewer gates than a circuit designed to give a precise result. Using approximate circuits in a noisy environment, the amount of noise is reduced which can potentially compensate for the precision loss of the approximation. We analyze some of the factors that, if present, can lead to a benefit from approximate circuits.

Finally, we showcase the NchooseK software package, which helps overcome the barrier to learning to program on a quantum computer. In the case of constraint based problems, especially constraint based optimization problems, NchooseK allows easy programming on multiple types of quantum computers. We provide evidence for decreased complexity in programming several NP-Hard problems, and we demonstrate the performance of NchooseK on the DWave Quantum Annealers and the IBM gate based machines and compare the results.

This work shows that even without direct access to quantum hardware, work can and is being done to make programming quantum computers both more accurate and more accessible.

Software Techniques for Improving the Accessibility and Performance of NISQ Era Quantum Computing

by
Ellis John Wilson

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Science

Raleigh, North Carolina
2024

APPROVED BY:

Huiyang Zhou	Jiajia Li
Jianqing Liu	Frank Mueller Chair of Advisory Committee

BIOGRAPHY

Ellis Wilson was born in Madison, Wisconsin. He attended what was then James Madison Memorial High School and stayed in Madison for his undergraduate degree in Engineering Physics at the University of Wisconsin-Madison. After a brief period working as a software engineer at Convergent Science, he began working on his Doctoral degree at North Carolina State University. During the course of his PhD work he also spent time as a summer intern at Berkeley National Lab and at Los Alamos National Lab.

ACKNOWLEDGEMENTS

Placeholder I would like to first thank my academic advisor, Frank Mueller. You have been extremely patient with me and provided much support and many connections over the course of my studies. You have also encouraged me to go outside and experience North Carolina.

I'd also like to thank Scott Pakin, who was my mentor for my three years working with Los Alamos National Lab. Thank you for taking time to chat with me and help me adapt to my time in a national lab.

A huge thank you to my partner, Aidan Combs. You have been behind me every step of the way, and helped me be a person. I wouldn't have made it without you.

Finally, I'd like to thank my parents. You've always supported me, even though I'm so bad at staying in touch.

Thank you all.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER

1

INTRODUCTION

1.1 Motivations

Quantum Computing is currently in its infancy. Scientists have been studying it mathematically and have developed theories and algorithms for decades — Richard Feynman proposed the development of quantum computers around forty years ago [Preskill, 2023] — but physical machines have only been around for a fraction of that time.

Because quantum computing is still a developing area, there are a number of issues preventing it from seeing widespread use. Ignoring matters such as cost to run and physical size, the largest issues are as follows:

1. Current machines produce inaccurate results. The "qubits", which store and modify the information in a quantum computer in a way similar to bits in a classical computer, are subject to many different forms of noise. This noise either causes the stored information to transition to a state that differs from what it should be in theory, or cause classical computing devices that extract information from the qubits to misinterpret their states.

2. Current machines are small in terms of the amount of information they can process at a time. A quantum chip with 5,000 qubits, such as the DWave Advantage systems [Boothby et al., 2021], are considered large. While they are not directly comparable, one would

be hard pressed to find a classical DRAM memory chip with fewer than 10 billion bits (approximately one GB).

3. Current methods of programming quantum computers are entirely unintelligible to people who are only familiar with programming classical computers. This is true to a much greater extent than switching from one programming language to another; the thought process is different, leading to a high barrier to entry.

Not much can be done to address point (2) in software (it is an inherent physical property of the machines). Yet, the other two issues can be at least mitigated through developing software techniques. This helps to bring us closer to the theorized advantages of quantum computing [Arute et al., 2019, Pednault et al., 2019, Boixo et al., 2018].

1.2 Background

In a way similar to how classical computing acts on bits, changing their states between 0 and 1 according to various instructions, quantum computing acts on qubits.

There are many different ways to think of a quantum program, but one of the most popular ones is as a "quantum circuit", as seen in Figure 1.1. To understand quantum circuits, a few other terms must also be understood.

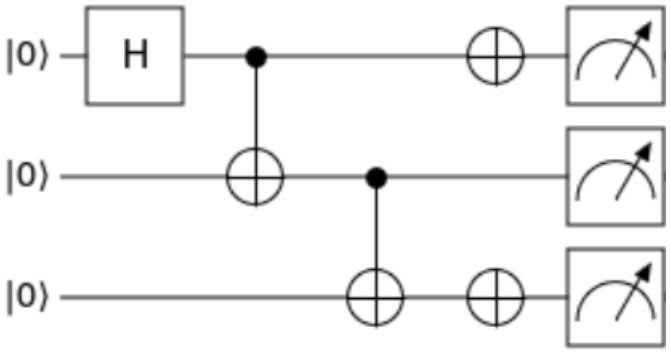


Figure 1.1: An example of a quantum circuit. Horizontal lines represent individual qubits, with symbols on those lines representing operations being performed on said qubits.

1.2.1 Terminology

Quantum State: Information is stored on qubits, which can occupy the state $|0\rangle$ (often thought of in the same way as 0 on a classical computer), $|1\rangle$, or anywhere in between. More generally, the single qubit state $|\psi\rangle$ follows the equation

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \tag{1.1}$$

where α and β are any complex numbers such that $\alpha\bar{\alpha}+\beta\bar{\beta}=1$

Superposition: When α and β are anything other than one or zero, the qubit is considered to be in a state of superposition. When such a qubit is measured, there is some probability it will be measured in the $|0\rangle$ state and some other probability it will be measured in the $|1\rangle$ state.

On a perfect system, $|0\rangle$ would be measured with probability $\alpha\bar{\alpha}$. Unfortunately, error is introduced when attempting to read the state of a qubit.

Because we can only measure these two states rather than directly measuring α and β, each circuit is often run multiple times to give an approximate α and β — and to mitigate the effects of readout error.

Quantum Gate: Quantum states are changed by way of operations referred to as quantum gates. A gate can change state $|\psi\rangle$ to any other single qubit state, but gates can also act on multiple qubits at the same time.

In practice, each time a gate is applied on a quantum device, some amount of error is introduced to the system. The exact amount of error depends on the system, the individual qubit, and on the gate.

Entanglement: Two qubits are considered to be entangled if the state of one qubit affects the state of another qubit.

One of the simplest states of entanglement to understand is called the Bell State. If two qubits are in the Bell state there is an equal chance of measuring both of them in the $|0\rangle$ state or both of them in the $|1\rangle$ state. A mixture of states cannot be measured in an error free environment.

Circuit Depth: The number of gates that must be operated sequentially on the critical path through a circuit is referred to as the circuit depth. Because each gate adds some amount of error and because the longer a qubit is active the more likely it is to develop errors on its own, circuit depth is one of the metrics for comparing circuit quality.

This depth is typically measured by the native gates of the system being used. Gates using more than two qubits, such as the three-qubit Toffoli gate [Toffoli, 1980], are compiled down to one and two qubit gates. Typically, even single qubit gates will also be compiled to a small set of native gates, which can be physically realized for a given particular system.

A shortcut is occasionally used to estimate circuit depth in a noisy environment by only considering the number of two-qubit gates rather than every gate, as the error introduced by two-qubit gates is approximately an order of magnitude larger than that of single qubit gates.

1.3 Contributions

In this work, we attempt to improve quantum computing accuracy and accessibility by way of altering the software available to the end user.

Hypothesis: We hypothesize that contemporary quantum computing devices require advances in noise reduction on both systemic and implementation levels combined with higher level program abstractions to provide computational utility that complements classical computing.

The contributions of this work are divided into three parts presented as chapters. Each addresses a part of this need of utility.

Chapter 2 seeks to demonstrate how the amount of noise can be reduced by determining and selecting the best performing qubits for the task at hand. In this chapter, we measure performance of several different benchmark algorithms at different times after the performance of individual qubits is measured. We show that assessing the quality qubits *closer to when circuits are run* allows us to select "better" qubits (in terms of lower noise) and thus obtain more accurate results.

In Chapter 3, we demonstrate the concept of *approximate* circuits for purposes of depth-reduction. We evaluate these approximate circuits under different circumstances and on different systems, and show that in the presence of noise and under certain conditions approximate circuits outperform longer, more precise circuits.

In Chapter, 4 we present the software package NchooseK, which allows easier programming of constraint based problems on quantum computers. We explain how NchooseK works, how it can be programmed, and then evaluate its performance on different types of quantum computer.

CHAPTER

2

JUST-IN-TIME QUANTUM CIRCUIT TRANSPILATION REDUCES NOISE

2.1 Introduction

Today's quantum computing devices and those of the foreseeable future are referred to as Noisy Intermediate Scale Quantum (NISQ) computers due to the noise inherent in the systems and the small number of quantum bits (qubits) available for calculations [Murali et al., 2019a]. Even when calculations can be performed with a small number of qubits, the noise in the quantum systems frequently produces incorrect results, which presents a challenge in using quantum computation. Consequently, techniques to identify, mitigate, and tolerate noise and even errors in calculations are of considerable importance for quantum computation amid this noisy reality.

Different types of errors can be distinguished. The most commonly reported errors are:

- Readout errors: These are errors in interpreting the state of the qubit at the end of the calculation, e.g., reading a qubit in the $|0\rangle$ state as being in the $|1\rangle$ state.
- Single qubit gate errors: These occur when applying gates to a single qubit causes small changes in the qubit state, which can accumulate over deep circuits with long sequences

of gates.

- Two-qubit gate errors: These result from interaction between two qubits under a two-qubit gate operation (e.g., both qubits of a CNOT gate);
- Decoherence errors: These are due to the decay of state over time in today's quantum devices, and they are referred to as T1, T2, and T2* — but are not addressed in this work.
- Cross-talk errors: These result when the state of a qubit or a resonator between qubits influences the state of another qubit or resonator in close vicinity — but are again beyond the scope of this work.

Each of these errors can vary from one qubit to another and also from one connection (coupling) to another; some qubits/connections experience less noise and fewer errors than others. To make matters more complicated, the qubits themselves change over time in an unpredictable fashion (due to the quantum nature of the qubit system), leading to a need to re-calibrate the qubits and recalculate these errors, e.g., once per day on IBM Q systems.

One way to reduce errors in quantum computations, especially on systems with more physical qubits than necessary for the circuit in question, is to try to map the circuit onto the most appropriate qubits during a process called "transpilation". Transpilation traditionally considers the mapping of logical qubits in a program onto a physical NISQ device with limited qubit connectivity and native gate operations. This may require a high-level gate (e.g., X/Y/Z rotation) to be translated into one or more low-level gates (e.g., U1/U2/U3 for IBM) with specific phase angles. Transpilation may result in logical qubits being moved (via swap operations) from one physical qubit to another throughout a circuit during its execution. More contemporary transpilation considers virtual-to-physical mappings to the highest fidelity qubits and connection between qubits to reduce the overall error [Tannu and Qureshi, 2019c, Murali et al., 2019a]. These optimizations are clearly non-trivial, as many mappings exist in this multi-dimensional non-linear optimization space. For example, the highest fidelity qubits for one circuit may not provide the connections for two-qubit gates of another circuit. It is therefore important to have accurate fidelity data of the physical machine for which a circuit is being transpiled. Different transpilers exist, each of which accept different types of statistical error values per qubit and per coupling between qubits before attempting to provide a high quality mapping. For IBM's quantum computers, these error metrics are derived from calibration runs of circuits that measure qubits and compare values with reference results. Such calibration occurs usually once per day, and error metrics are published on IBM's websites and can also be obtained from the Qiskit API [Aleksandrowicz et al., 2019] for the latest calibration run.

We have performed a series of experiments that, after an initial stable phase, uncovered a quick deterioration in fidelity of qubit gates, measurements and couplings not too long after calibration. These experiments included micro-benchmarks to (a) prepare an n qubit circuit with initial state $|0\rangle^n$ followed by simply measuring each qubit and (b) subjecting each qubit to a series of X (NOT) gates before measurement. When repeated hourly, no clear trend became visible. Neither could we detect when the original calibration took place, nor did we observe a gradual de-calibration. When employing longer and more complex circuits and directly comparing different error values, we found that while some qubits remained stable, other qubits showed significant variations in fidelity throughout the day.

These findings motivated us to experiment with obtaining calibration data ourselves, use them in just-in-time transpilation, and then observe errors for this transpiled circuit compared to one transpiled with IBM's calibration data. Clearly, if virtual-to-physical mappings differ between just-in-time transpilation vs. default transpilation, the fidelity of results can be expected to differ as well.

We chose to focus the investigation on readout errors and two-qubit gate errors in particular, the most significant errors in magnitude. This is also motivated by readout errors affecting every circuit and two-qubit gate errors consistently being about an order of magnitude higher than single qubit gate errors, i.e., the qubit placement of two-qubit gates during transpilation is of high importance for overall fidelity.

Readout errors were determined by immediately measuring a newly prepared qubit and subsequently applying a single X (NOT) gate before measuring the same qubit again. We compared the results to the expected values (of all $|0\rangle$ or all $|1\rangle$, respectively) to obtain the percent error. Two-qubit gate errors were determined by utilizing IBM's built-in randomized benchmarking capability to obtain an error value. We then subjected transpilation to our error values instead of the default IBM ones (from daily calibration). This resulted in different qubit mappings leading to an improvement of 3-304% on average, and up to 400%.

2.2 Design

The design of our just-in-time transpilation was driven by an initial experiment followed by a methodology to address shortcomings of the current system. While observations are specific to IBM Q devices, the methodological approach is more generic and may transfer to other NISQ devices.

2.2.1 Motivating Experiments

Our first objective was to determine whether or not the fidelity of qubits varied significantly throughout the day. If the fidelity of qubits did not vary significantly between two calibration instances, any effort to repeatedly assess the error rates would likely not contribute to fidelity improvements. To test our hypothesis of variations, we conducted experiments with a number of circuits assessing reported errors throughout the day.

The main focus centered on the qualitative aspect of qubit change, i.e., do qubits provide different results in fidelity over time between calibrations, rather than absolute errors. To this end, experiments were limited to simple circuits to assess readout errors or errors due to successive Pauli gates, which were repeated every hour.

The first experiment focused on readout errors without any gates, where a qubit was initialized ($|0\rangle$ state) and then immediately measured. The second experiment assessed readouts for a qubit after a Pauli X (NOT) gate., i.e., the $|1\rangle$ state. Fig. 2.1 depicts hourly measurements (x-axis) over the percentage of correct results (y-axis) on different days (colored data series) in 2019 on the IBM *Poughkeepsie* device (20 qubits). The results show that qubits do not remain stable between calibrations. This behavior was observed across qubits and different IBM Q devices.

While readout results of these circuits usually did not change drastically from hour to hour, they did change in an unpredictable manner, sometimes resulting in better accuracy, sometimes in worse. This made it impossible to infer or reverse engineer when the calibration actually took place, i.e, we did not observe a drastic change in quality for qubits when measured. We also tested circuits with many gates in a less rigorous manner and observed similar results. Based on prior work, we know that circuits with virtual qubits were mapped onto physical ones via transpilation. Using IBM's optimization level 3 (the highest level at the time of this writing) lets one take IBM's error data from the last calibration into account [Murali et al., 2019a, Tannu and Qureshi, 2019c]. This led us to the new hypothesis that, when selecting physical qubits to which circuits are to be mapped, a new set of error measurements for just-in-time transpilation might improve the overall fidelity.

2.2.2 Error Selection

A number of different types of errors are taken into account when mapping circuits to qubits, where some of these errors are more prevalent than others as indicated by the respective error metrics. For example, T1 and T2 errors are significant in long circuits but not in short ones. Gate errors will be present in all circuits, but more so in long circuits using many gates. Readout errors need to be taken into account in all circuits. If some errors are more significant than

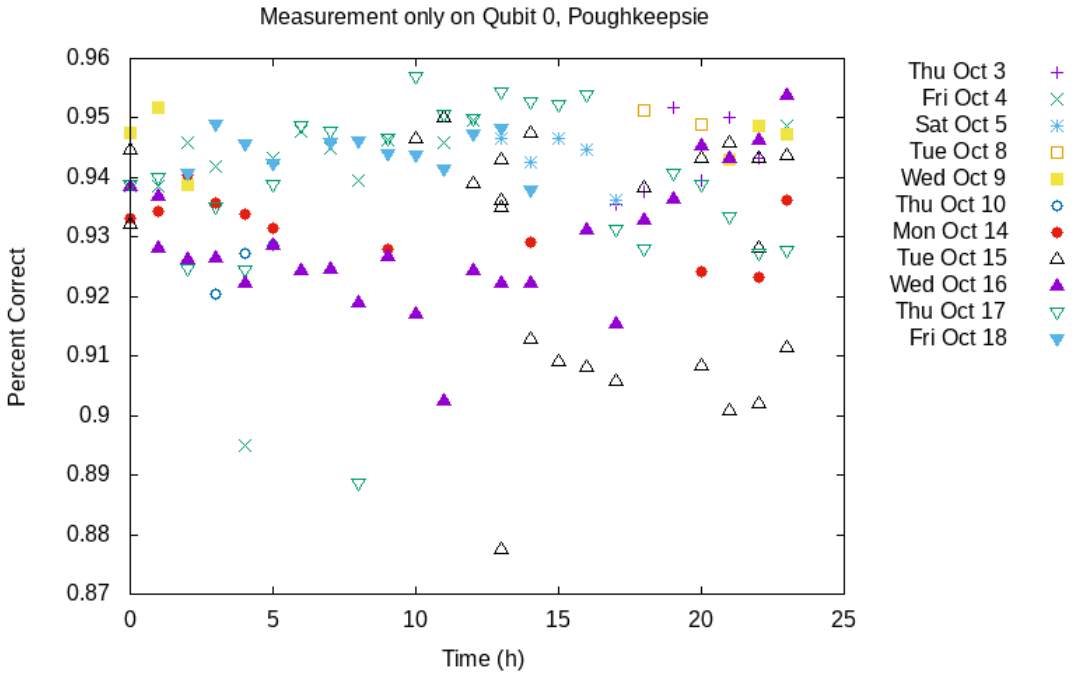


Figure 2.1: *Measurements with no gates on qubit 0 of IBM Poughkeepsie over time. The fidelity of readouts for the qubit varies in a chaotic (non-predictable) manner. Results for other qubits and* $|1\rangle$ *circuits are similar, figures omitted due to space.*

others, those errors dominate the mapping decisions, while other, less significant ones will only marginally contribute.

We decided to focus on two types of errors, those from readouts and those from two-qubit gates. Readout errors affect any circuit, and their probability is relatively high on today's NISQ computers. Readout errors are reported to be in the order of 10^{-2} for IBM Q devices. We even observed that sometimes they can be as high as 10%.

We also focus on two-qubit gate errors for the same reason: They have an equally high error rate (both reported by IBM and observed by us). In contrast, single-qubit gate errors are reported to be lower (10^{-3} for IBM Q devices), and they were also an order of magnitude smaller than readout or two-qubit errors in our experiments.

2.2.3 Methodological Error Collection

The challenge at hand is to reliably collect error characteristics of a physical quantum device that can subsequently be used to map circuits to physical qubits such that overall fidelity can be increased. Readout errors are the easiest to be measured, simply by constructing a circuit that minimizes any of the other types of errors while producing a known measurement value. To

minimize gate and time-based errors, the qubits are measured as quickly as possible and with the fewest number of gates. We observe that reading $|0\rangle$ and $|1\rangle$ states each have different error rates [Tannu and Qureshi, 2019a, Tannu and Qureshi, 2019b]. Hence, we utilize two circuits per qubit to characterize readout errors. The first circuit prepares a qubit in the $|0\rangle$ state (as quickly as possible) and measures it, while the second one prepares the qubit in the $|1\rangle$ state via a single X (NOT) gate before measuring.

Gate errors present more of a challenge to be assessed. Recall that we focus on two-qubit gates here due to their higher error rates compared to single qubit gates. Two-qubit gate errors are determined for each pair of connected qubits that can be captured through randomized benchmarking, which uses randomized sequences of gates of increasing length resulting in a known $|0\rangle$ state on qubits. By comparing actual measurements to this known value, error rates are determined. This is described in more detail in [Magesan et al., 2011].

These error characteristics are subsequently used for just-in-time transpilation of circuits for mapping to physical qubits with high fidelity for couplings/connections within the circuit and high measurement quality of selected qubits.

2.3 Implementation

We decided to implement our high-level design of just-in-time transpilation for IBM Q devices using Qiskit. This involves data collection on errors on an IBM Q device, subsequent transpilation of benchmarks via Qiskit at an optimization level that takes errors into account when mapping to physical qubits, and running these benchmarks on the same IBM Q device.

In order to test whether just-in-time error measurement improves performance over using the daily calibrations, we need to to reliably collect data on errors and, for a fair comparison, in a similar manner to how IBM collects data and reports errors during their daily calibrations.

Due to the nature of IBM's qubits (and other technologies as well), the error for reading a qubit in the $|0\rangle$ ground state state is much lower than reading a qubit in the $|1\rangle$ excited state, which is less stable [Tannu and Qureshi, 2019a, Tannu and Qureshi, 2019b]. IBM determines readout error rates for each state as well as the average of both, which it reports as the readout error. These errors are relatively easily obtained. As described in our motivating experiments, to assess errors for readouts of the $|0\rangle$ state one merely needs to measure immediately after preparing a qubit. Similarly, the $|1\rangle$ state is read out after a qubit is prepared and subjected to a single X (NOT) gate. The observed level of error between the single qubit gate and the readout error, especially when in the $|1\rangle$ state, shows that the contribution of the X gate to the error is negligible (about an order of magnitude lower than the readout error). The readout error is this calculated as the percent of incorrect results returned from the respective circuits.

The two-qubit error requires more complex circuits. We employ Qiskit's randomized benchmarking capabilities, which can automate the process of data collection. These randomized benchmarks consist of circuits with two qubits that are generated such that their output is an "Error per Clifford" value, which is proportional to the two-qubit error itself.

Obtaining these error metrics for each qubit is a computationally intensive task. Due to limited compute cycles, we decided to combine many of the individual qubit measurements into a single multi-qubit measurement. While this ignores the impact of qubit crosstalk, it still remains useful as any circuit, including our benchmarks, also utilizes multiple qubits, often in close physical vicinity to reduce the number of swaps in transpiled programs. We split the two-qubit gate errors up such that only one coupler of a given qubit was assessed in terms of error at a time. On the IBM Q devices used here, the maximum degree of a qubit is three, i.e., we ran a total of three jobs to capture all two-qubit errors. Once each of these errors had been measured, we assessed the virtual-to-physical mappings. This allowed us to report errors for each physical qubit.

As we are focusing on IBM Q devices, we decided to leverage Qiskit's built in transpiler using the highest optimization level available (level 3) in order to trigger an optimization for virtual-to-physical mappings [Murali et al., 2019a].

2.4 Experimental Setup

We conducted experiments on various IBM Q devices throughout different days and different times as well as repeatedly during a particular day. In every experiment, we first manually measured the CNOT and readout errors and then, based on this error information, transpiled our circuits before sending them to the devices to execute them. We kept track of the circuit layouts post-transpilation and their performance with respect to accuracy. Next, we describe the individual aspects of this setup.

2.4.1 Device Information

We performed our experiments primarily on two IBM Q devices, *Almaden* (20 qubits) and *Paris* (27 qubits). The rationale was to select backends with a sufficient number of possible virtual-to-physical qubit mappings so that the transpilation procedure could adapt mappings to error data. Both devices allow a total of 900 circuits to be sent in one job. Availability of these devices presents another challenge, as they tend to be busy with many jobs in the queue, which meant that the calibration job was running an hour or more before the benchmark jobs as the latter can only be submitted after transpilation taking errors from the former job

into account. This assesses just-in-time transpilation in a normal user scenario with so-called "fairshare" queuing. In addition, we conducted experiments in "dedicated" mode, available only to select users, where calibration and benchmark jobs can be run within minutes of each other, again after transpilation of benchmarks based on error data from immediately preceding calibration. Figure 2.2 depicts the physical qubit topology of the two backends, Almaden and Paris, with a snapshot of the calibration-of-the-day (COTD) data encoded as colors according to the respective heatmap of the device.

2.4.2 Benchmarking Circuits

We selected a number of circuits for just-in-time transpilation also used in prior work [Murali et al., 2019a]. The characteristics of the selected benchmarks were based on the ability to scale single qubit gates, two-qubit gates, circuit depth and circuit width (i.e., the number of qubits). These benchmarks can be parametrized by the number of qubits, n, and are:

1. bv(n): the Bernstein-Vazirani algorithm that learns an n-bit string encoded in a function and reads out $n+1$ qubits;
2. hs(n): the n-bit/qubit hidden-shift algorithm that determines the constant by which the input of one function is increased (shifted) relative to that of another function, where n qubits are measured;
3. qft(n): the n-bit/qubit quantum Fourier transform algorithm, which is used in many other quantum algorithms as a building block with n qubits measured;
4. toffoli(n): the n-qubit "universal" Toffoli gate that can be specialized for a number of arithmetic operations depending on parameters, $n+1$ qubits are read out;
5. adder(n): an n-bit adder algorithm using $2\times n+2$ qubits and $n+1$ readouts.

Algorithms 1-3 include Hadamard gates and conditional rotational gates, yet still have known reference outputs. Conversely, algorithms 4-5 consist of Pauli gates, C-NOT gates or CC-NOT gates (with two conditionals), the latter of which can be transpiled into a sequence of single qubit (Hadamard and rotational) gates and six C-NOT gates, again with known expected outputs.

2.4.3 Qiskit Experiments

Qiskit provides an interface for sending multiple quantum circuit experiments to the device in a single job. The maximum number of these experiments depends on the type of device.

As an example, the *Almaden* device accepts a total of 900 circuits in a single job. CNOT and readout error calibrations are performed in our experiments using the calibration circuits detailed in previous section. These are run repeatedly at a particular time of the day, as shown in Figure 2.3 (timestamp1, timestamp2 etc.). With the resulting error data, several benchmark circuits are transpiled. We investigate 5 circuits representing the above benchmark codes per run, where each circuit is executed for 4096 shots, i.e., repeated circuit executions with a measurement. Further, we have 3 sets of these benchmarks with increasing number of qubits as shown in the figure (low/medium/high number of qubits) in a single job. In total, a single benchmark measurement job contains 75 circuits, i.e., 25 circuits per benchmark set and 5 circuits for each individual benchmark with exactly the same circuit and mapping since they are transpiled together with the same calibration data using the *qiskit.compiler.transpile* function. We execute several circuits for a particular benchmark, each with 4096 shots, as we observed that for certain qubits and connections significant variations in the accuracy exist across different circuit executions within the same job. In summary, each benchmark job that gets submitted to the device is just-in-time transpiled with the latest error data obtained by our measurements — instead of the default COTD data from IBM. Depending on the experiment, these jobs are either run at different times on different days in "fairshare" queuing, or they are repeatedly run throughout the day in "dedicated" time slots to capture the variance in the accuracy of benchmark circuits. Notice that dedicated execution is a novel feature that became available only in late May 2020.

2.5 Results

We first report results in the default user mode, followed by dedicated mode. We then perform a sensitivity analysis with respect to circuit layouts before discussing overall findings and implications.

2.5.1 Fairshare User Mode

These experiments on IBM Q *Almaden* consist of a first job running all benchmarks resulting from level 3 transpilation using IBM's error data, followed by eight instances of two jobs, one for measurement to obtain refreshed error data and a second to run all benchmarks transpiled at level 3 with the fresh error data. Percent accuracy relative to expected results (y-axis) is plotted for each benchmark run (x-axis).

Figure 2.4 depicts results for Hidden Shift (hs) with 4, 6, and 8 qubits, where the x-axis indicates the time (during the day) when the benchmark run started and the number of minutes

prior to which error data was measured.

For hs(4), the top graph in the figure, the first data point shows an accuracy of 47% (with a small standard deviation indicated by the whiskers) for IBM's reference calibration 10 hours earlier. This is the reference run (dashed line) for our experiments. The remaining data points are showing 67-73% accuracy for our runs, spaced in 1-2 hour intervals whenever the job queue scheduled runs, with prior error data obtained 39 minutes to nearly 2 hours earlier. The different colors of data points indicate distinct layouts of virtual-to-physical qubits. Our layouts differ from IBM's layout due to the refreshed error data, which provides the benefits in accuracy.

For hs(6) and hs(8) in middle and lower graphs of Figure 2.4, our results have an even higher improvement in accuracy over IBM's reference layout, with our layouts changing from hour to hour. Overall, IBM's accuracy is reduced from 47% to 27% to 12% for hs(4), hs(6) and hs(8), respectively. This reflects the higher number of qubits used and longer depth of a given circuit. With our just-in-time transpilation, the values are much higher: 70%, 58%, and 45% on an average for hs(4), hs(6) and hs(8), respectively.

Results for other benchmarks are similar in trend, albeit with different absolute accuracies/improvements with figures omitted due to space. Relative improvement in accuracy ranges from 8-48% for bv, 48-304% for hs, 45-69% for qft, 133-155% for toffoli, and 12-42% for adder, with maximum improvements sometimes as high as 400%, i.e., a factor of four improvement in accuracy. We also observe that toffoli, qft and adder have a higher standard deviation.

Observation 1: Just-in-time transpilation tends to improve the relative accuracy of measured results on average by 8%-304% and up to 400% in extreme cases in fairshare user mode, with smaller benefits for smaller circuits, with high fidelity and larger benefits for large circuits with low fidelity. Best layouts change at least hourly.

Figure 2.5 depicts results for Bernstein-Vazirani (bv) with 4 qubits on two different days. On the first day (upper graph), trends are similar to hidden-shift, where the accuracy of just-in-time transpiled benchmarks throughout the day is consistently higher (around 82%) than those transpiled with using IBM's COTD (69.5%). The difference between our measurements is relatively small (+/-5%). But on a different day (lower figure), results are mixed as the benchmarks transpiled with IBM's COTD show higher accuracy (83%) while many of our just-in-time transpilations result in lower accuracy (as little as 76%) while others are slightly better (up to 85%) than IBM's reference. Interestingly, all the benchmarks show more significant standard deviations (wider whiskers) in the lower graph, even though IBM's calibration was about 10 hours prior in both cases. Closer inspection reveals that the same IBM layout mapping (blue dot) also provides slightly better results (3rd and 9th data point), yet worse results at a different time (8th data point).

Observation 2: Benefits of just-in-time transpilation vary from day to day, even for the same layouts of qubits on a device.

While we observe such variation, we actually cannot provide absolute conclusions from this data as we only ran benchmarks with IBM's COTD layout once, and only hours apart from our just-in-time experiments. This led us to conduct a set of experiments in dedicated mode close together in time, once this mode became available. This is discussed in the next subsection.

The QFT circuit (figures omitted due to space) contains a large number of two-qubit gates and thus results in lower overall accuracy and also declining accuracy as circuits are scaled up from 4 over 6 to 8 qubits. As before, IBM's accuracy is generally lower than ours (30% vs. 18% for 4 qubits) but the total value becomes unreasonably low for 8 qubits (IBM: 0.85%, ours: 1.3%), even though our results are still better on one of the days. However, on another day, only half of our just-in-time calibrations resulted in benefits over IBM's, still with the same low accuracy under qubit scaling. The Toffoli and adder benchmarks show trends similar to the QFT benchmark.

Observation 3: As the number of qubits is scaled up, total accuracy drops significantly to the point where few results remain correct, even with just-in-time transpilation. IBM's results remain inferior to our just-in-time method.

2.5.2 Dedicated Mode

In regular user mode, fairshare queuing [IBM, 2020] on IBM Q devices prevents a calibration job to be run back-to-back with benchmarks as just-in-time transpilation requires the error data from the calibration run, and typical queue delay is on the order of hours for IBM Q Hub devices (or even days for public devices). While we showed that qubit fidelity in terms of readout and coupling errors varies, our prior results were inconclusive with respect to the rate at which these variations take place.

A novel dedicated queuing mode allows the reservation of time slots of fixed lengths at a given time of the day. This allows us to reserve a slot long enough to run a calibration test to obtain readout and coupling errors, run benchmarks using IBM's error data while transpiling with our newly obtained error data, and then run the just-in-time transpiled benchmarks based on our error data. These three jobs run back-to-back within 15 minutes. This experiment was repeated 8 times during a 24-hour period. Dedicated queuing was available on IBM's *Paris* device with 27 qubits.

Figure 2.6 depicts the accuracy for a 2+2, 3+3 and a 4+4 adder (upper/middle/lower graphs) in dedicated mode. Black dots indicate IBM's layout based on their COTD errors obtained 28-49 hours earlier. Notice that the device was *not* recalibrated during this period, which indicates

that IBM even calibrates less frequently than the 24 hours that are commonly cited. Each set of (black, colored) data points runs within the same time slot and should be related to one another in comparisons.

For the first graph, we observe that IBM runs (black) vary significantly in accuracy over time, as much as 29-37%, i.e., a given calibration with COTD error data does not provide consistent results. We further observe that when any IBM run (black) is followed by our just-in-time transpiled run (colored) minutes later, the latter always provides higher accuracy. Standard variations are sometimes higher, sometimes lower with no clear pattern. As circuit sizes are scaled up (middle/lower graphs), this trend still holds, even as absolute accuracy becomes smaller due to wider and deeper circuits. The benefits of just-in-time transpilation are more pronounced in the 3+3 adder (middle graph), without any clear cause as these three benchmarks ran back-to-back (cf. absolute times indicated on the x-axis). Just-in-time transpilation always resulted in a different circuit than IBM's default transpilation, and the former resulted in notable savings — with the one exception of adder(4) in the 2nd to last pair of (black, yellow) dots, where our benefit is smaller. Layouts change between hourly slots. These results generalize to other benchmarks with higher (bv, hs) or lower (qft, toffoli) absolute savings. We did see occasional outliers as discussed in the next subsection. We summarize these findings as the following observation.

Observation 4: Just-in-time transpilation offers more significant benefits when error data is obtained immediately prior to an application circuit, irrespective of circuit width and depth.

We also conducted a sequence of experiments in a single 1-hour slot, where the IBM-transpiled benchmark was run back-to-back with four instances of (a) re-calibration (obtaining fresh error data) used by just-in-time transpilation followed by (b) executing all benchmarks. Our method was superior in all cases except for adder(4), qft(6), toffoli(3), and sometimes better/sometimes worse for qft(8).

Observation 5: Even when error data is obtained immediately prior to an application run, just-in-time transpilation cannot *always* guarantee to provide superior results. Variations are more pronounced long-term but also exist to a smaller extent short-term. Best layouts change even within minutes.

2.5.3 Detailed Accuracy Improvement for Dedicated Mode

Figure 2.7 depicts the average percent improvement in accuracy for dedicated benchmark runs on the IBM Paris device normalized to just-in-time transpilation with IBM's transpilation as a baseline. Each bar corresponds to a separate run in a dedicated time slot over a 24-hour period,

i.e., 8 time slots in total. Different colors indicate different mappings.

Overall, most cases show a moderate to significant improvement with the occasional exception of an insignificant loss (a few instances of qft(4) and qft(8)) and few more significant losses (one instance each for hs(6), hs(8), toffoli(3), toffoli(4)). In terms of absolute accuracy, there was one data point for hs(6) where IBM's result (59%) was better than ours (52%), and another in hs(8) with 52% vs. 44% within the same benchmark run. We do not have an explanation as neither hs(4) nor any other benchmark in the same run showed inferiority of our method. The same holds for the last run for toffoli(3) and toffoli(4). All these outliers have in common that they use a never-seen-before layout, which may indicate that the error collection method could possibly be improved on.

The overall average in improvement (over all 8 runs) is indicated by a dashed line.

Observation 6: Just-in-time transpilation tends to improve the relative accuracy of measured results on average by 3%-190% and up to 150% in extreme cases in dedicated mode, again with high fidelity and larger benefits for large circuits with low fidelity. Best layouts change within minutes.

In summary, Figure 2.7 reinforces the last two observations in that just-in-time transpilation provides benefits in the majority of cases, but there are exceptions.

2.5.4 Circuit Layout Analysis

Results so far have shown that differences in accuracy are correlated to just-in-time transpilation on recent error data. We investigated the benefits in a sensitivity study by considering changes in virtual-to-physical qubit mappings. To this end, the resulting virtual layouts were superimposed on the heatmap-coded interconnect of a quantum device. Figure 2.8 depicts pairs of IBM/our layouts for hs(8) and adder(4). The nodes are qubits and edges are couplings. A solidly colored qubit indicates that this qubit is used within the respective circuit. Heatmaps range from low errors (green) over blue to high errors (red) on a scale indicated for each graph, i.e., separately for per-qubit readouts and couplings.

Overall, we can compare the errors of the IBM model (left) with that of our error data (right) agnostic of any circuit. The error values differ for a number of qubits and couplings, most notably couplings 4-7, 6-7, 5-8, and 12-15, and also qubits 0, 4, 5, 8, 15 and 17. Others are constantly good (many qubits and couplings remain green on both sides) or constantly bad (e.g., qubit 21).

In the adder(4) example, our layout provides worse accuracy than IBM's. First, we observe that in Figure 2.8a coupling 4-7 within the circuit has high errors (red), and qubits 5 and 8 have mediocre fidelity (blue/purple). In contrast, all couplings in Figure 2.8d are of higher

fidelity (green) while only qubit 8 has lower fidelity (purple). Yet, IBM's accuracy at 10% is better than ours at 8%. Closer inspection reveals that our lower end of the error spectrum has twice the error value of IBM's lower end errors for both readouts and connectors. This means the color spectrum on the right side should be shifted toward higher errors. Another significant difference is in the readout qubits, which are 4,7,8,9,11 for IBM's and 8,12,13,20,22 for our transpiled code. This accounts in part of the difference in accuracy, as will be discussed in the next subsection.

In the hs(8) example, our layout provides better accuracy than IBM's. We observe that the selected qubits and couplings for the circuit appear nearly equally good in Figure 2.8c and Figure 2.8d, with a slight bias to higher fidelity (lighter green) on the right side for qubits. As all qubits are read out, this could explain the difference, even after taking into account the differences in heatmap encoding. Notice that the hidden shift algorithm requires only pairs of two qubits to be coupled, which explains the layouts of isolated qubit pairs.

Observation 7: Differences in layouts corroborate the hypothesis that there are two classes of errors: "Persistent" errors due to low fidelity qubits and couplings that retain high errors, and "transient" errors that vary over shorter times. However, detailed analysis of layouts with respect to noise levels of qubit readouts and connectors remain only partially conclusive.

2.5.5 Discussion

The detailed analysis of layouts did not provide the clarity on a case-by-case basis that we had anticipated. It is possible that other factors have to be accounted for to explain differences in accuracy. In particular, it would be important to compare IBM's codes for determining error rates with ours as we see much higher rates. This could be due to the fact that the last calibration occurred hours ago, or it could indicate that our algorithms are more suitable to find good layouts. Furthermore, cross talk error is known to be in the order of readout and coupling errors. Single qubit gate errors are said to be an order of magnitude lower, as also reported by IBM after each calibration. Another factor is the number of times a coupling is used in conditional gates (e.g., CNOT) and, to a lesser extent, the number of single qubit gates. While we saw "permanently" high qubit and coupling errors for a few device elements, most of them either remain at higher fidelity or change in the medium range over time. It may be possible to further distinguish errors within time ranges of minutes vs. hours, but we do not have sufficient data to reliably do so.

With the results shown above, we conclude that dynamic on-the-fly error calibration helps in taking into account the current state of the qubits. Transpiling the circuits just-in-time with

this error information statistically produces more accurate results than those produced when the IBM's published error information is used to transpile the same circuits.

This work has the following implications: Our approach can help in producing better results in circuits from a statistical perspective, but does not eliminate errors.

Recommendation 1: We suggest to first obtain fresh error data from a device before running sensitive circuits.

This is easily done in IBM's dedicated mode but even provides benefits when hours lie in between obtaining error data and running the just-in-time transpiled circuit. An ensemble of circuits could then be prepared by transpiling with the dynamically measured readout error information, measured CNOT information, or both.

Recommendation 2: A circuit transpiled with the default error data should be included in experiments.

Sometimes, IBM's layout is superior, and a diversity of mappings can provide more accurate results [Tannu and Qureshi, 2019a].

Recommendation 3: Devices should either be recalibrated more frequently, or their errors should be assessed more often (possibly both), with results made accessible to users.

If users always prefaced their code with a fresh error data analysis, less science could be performed on a quantum device, yet results may be of higher value. This is a subtle conundrum, and the frequency of recalibration should be revisited by quantum backend operators.

Currently, the device properties (*backend.properties()* object) contain the calibration information published by IBM earlier during the day. We suggest that IBM also publish more dynamic error information along with the accuracy of the circuits by periodically running these error extraction circuits. The developers could then, based on the accuracy of results, decide whether or not to exploit this dynamic error information to transpile their circuits or call their own error measurement jobs. Furthermore, noise-based transpilation (level 3 in Qiskit) should be the default. Finally, job dependencies and server-side transpilation should be introduced in fairshare user mode to allow a second job to be transpiled depending on output data of the first job that ran minutes before.

2.6 Related Work

Current NISQ machines require substantial tuning of control signals in order to compensate for noise in individual devices. The closest related work to ours focused on noise-aware mappings and read-out errors [Murali et al., 2019a], which is using noise data to adapt qubit mappings during the transpilation process. This technique was later integrated into IBM's Qiskit transpilation, which uses daily calibrations for qubit mappings. As our work shows, more frequent

noise recalibrations provide additional benefits on today's NISQ devices.

Other techniques focus on interpreting different qubit mappings statistically and inverting computational results to benefit from lower errors in non-excited states [Tannu and Qureshi, 2019a, Tannu and Qureshi, 2019b], hardware-specific optimizations confined to back-end passes of the compiler across different NISQ platforms [Murali et al., 2019b], or reduction in cross talk [Murali et al., 2020]. IBM uses pulses to further reduce noise [Bishop and Gambetta, 2019], a technique that was generalized to larger circuits or blocks of gates with shorter pulses [Shi et al., 2019, Gokhale et al., 2019].

Other qubit mapping approaches were shown to be effective for smaller-scale NISQ devices [Murali et al., 2019a, Zulehner et al., 2018, Tannu and Qureshi, 2019c] but often required high time/memory consumption when scaling up, while others had more scalable algorithms but compromised in the fidelity of the mapping [Wille et al., 2016, Siraichi et al., 2018], while yet others focused on scalability without considering noise details at the same level of detail [Li et al., 2019], or used dynamic assertions as a means to filter by noise [Liu et al., 2020]. These techniques can orthogonally improve results on top of our recalibration.

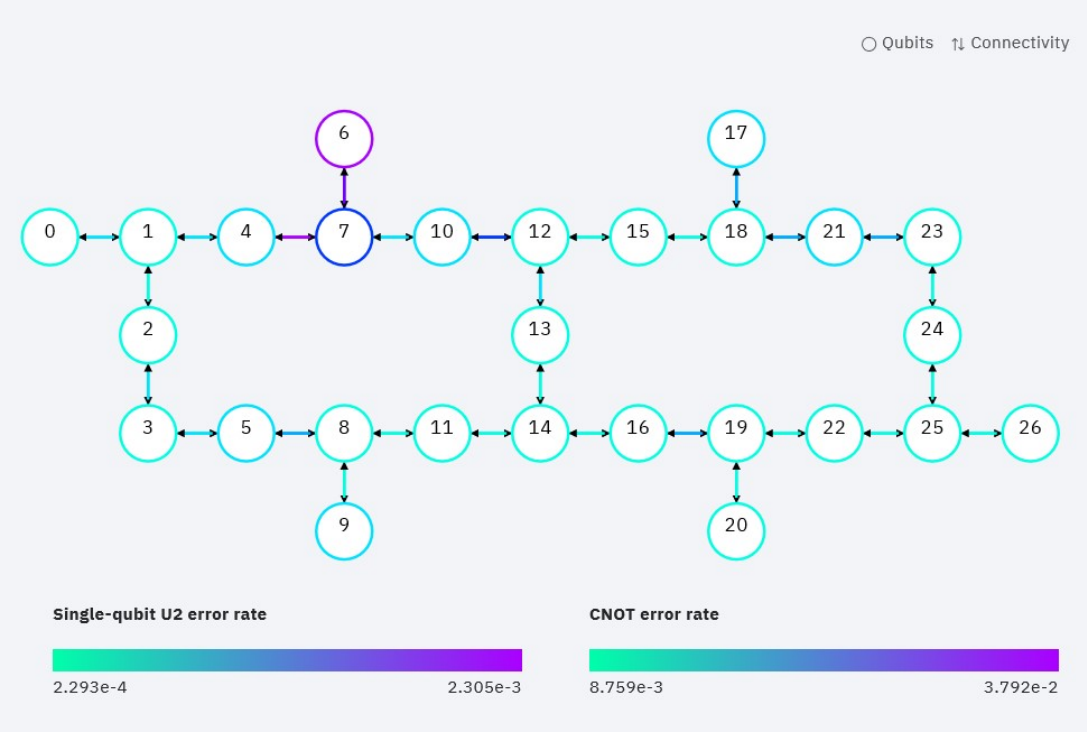
2.7 Conclusion

We have contributed a methodology for on-the-fly transpilation taking fresh error data for readouts and two-qubit gates into account. Our experiments have shown the effectiveness of this technique on current NISQ devices resulting in 3-190% improvement of accuracy for dedicated execution and 8-304% for shared job queues with a maximum observed improvement of a factor of four, depending on the circuit. Improvements are best when error data was recently obtained, leading to recommendations for adjusting operations of quantum devices to obtain and publish error data more frequently.

Acknowledgment

We would like to thank Ali Javadi-Abhari for pointing out problems with calibrations. Access to the IBM Q Network was obtained through the IBM Q Hub at NC State. This work was funded in part by NSF grant 1917383.

(a) IBM Q Almaden device



(b) IBM Q Paris device

Figure 2.2: *Qubit connectivity with colored mappings for qubits and connections corresponding to COTD information on a heatmap range.* Source:

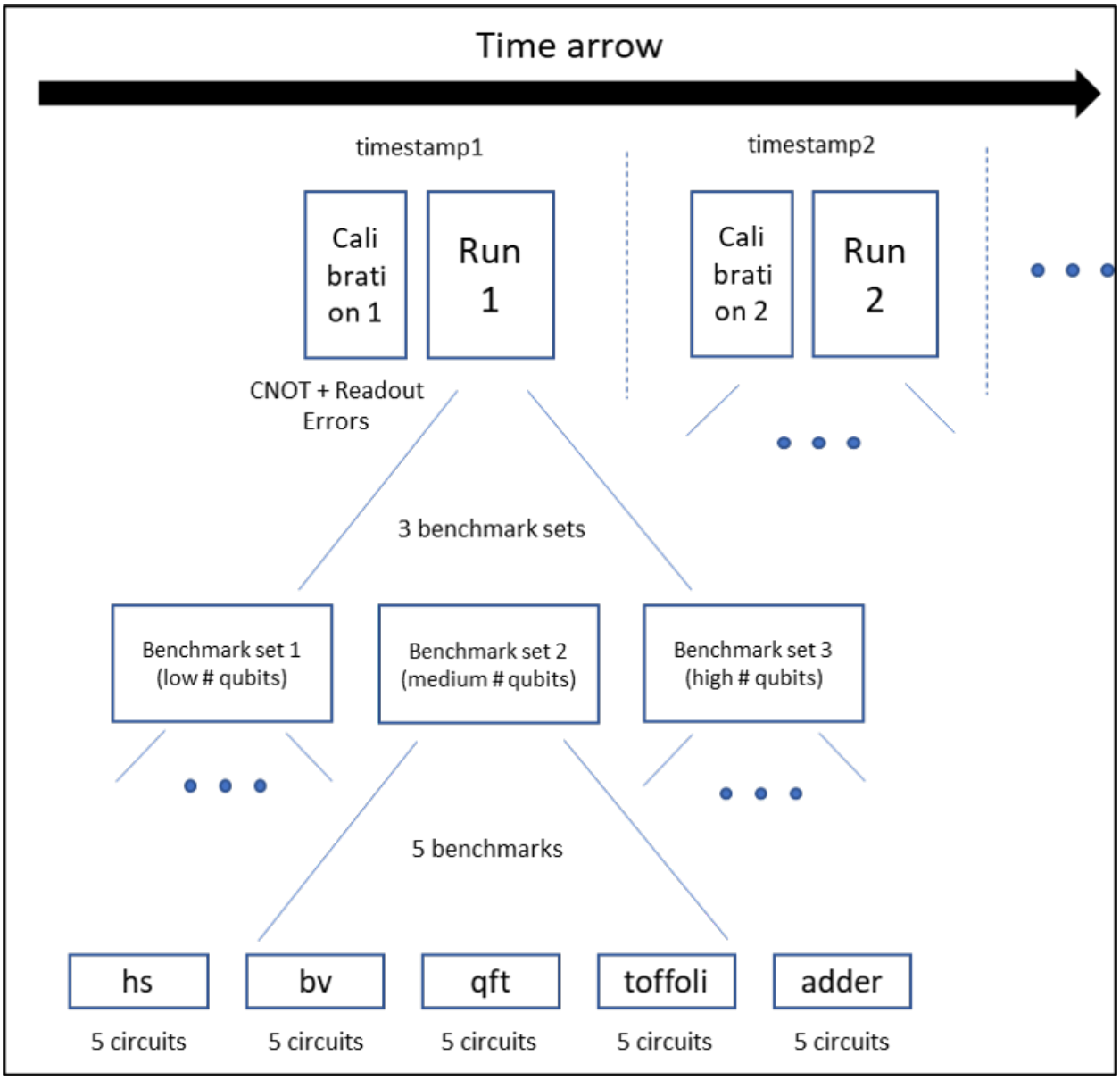


Figure 2.3: *Calibration and benchmark circuits as arranged in our job framework*

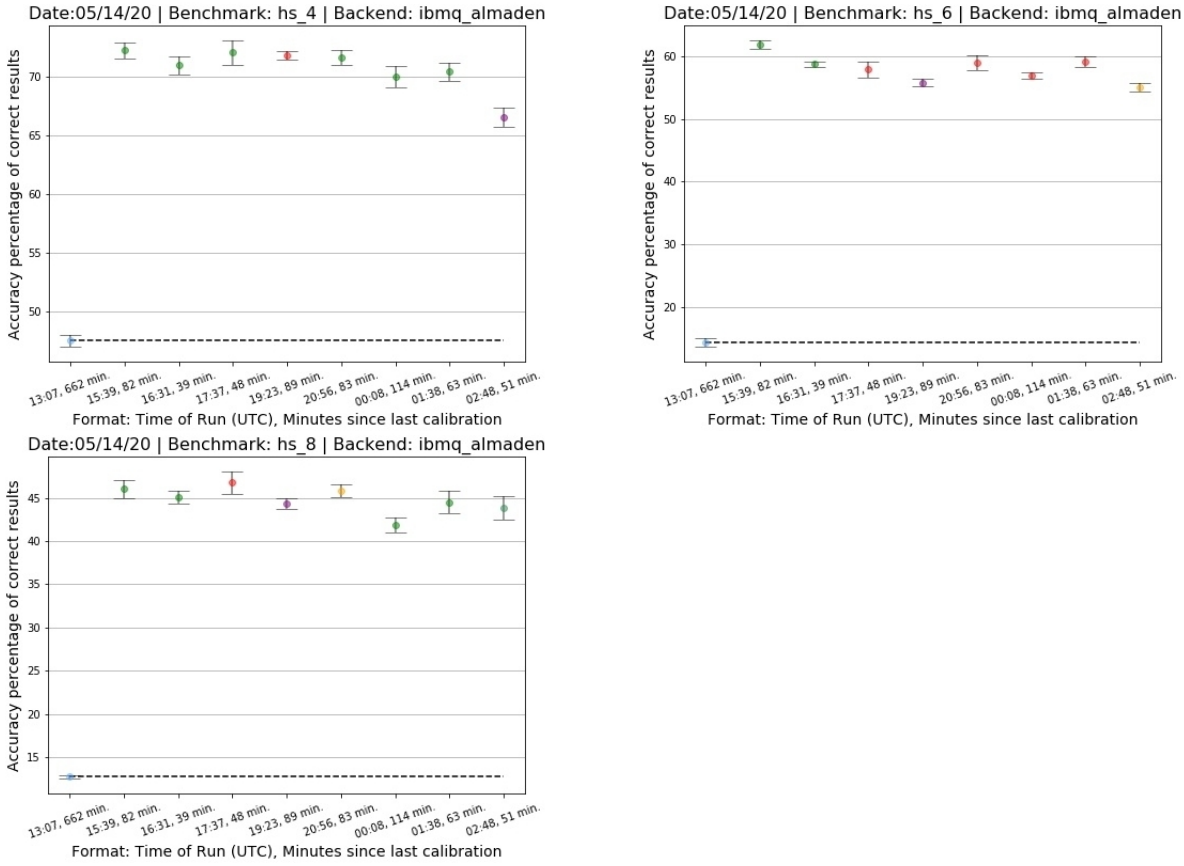


Figure 2.4: *Accuracy of Hidden Shift for 4/6/8 qubits (upper/middle/lower graphs) at different times during the day with prior calibration in minutes.*

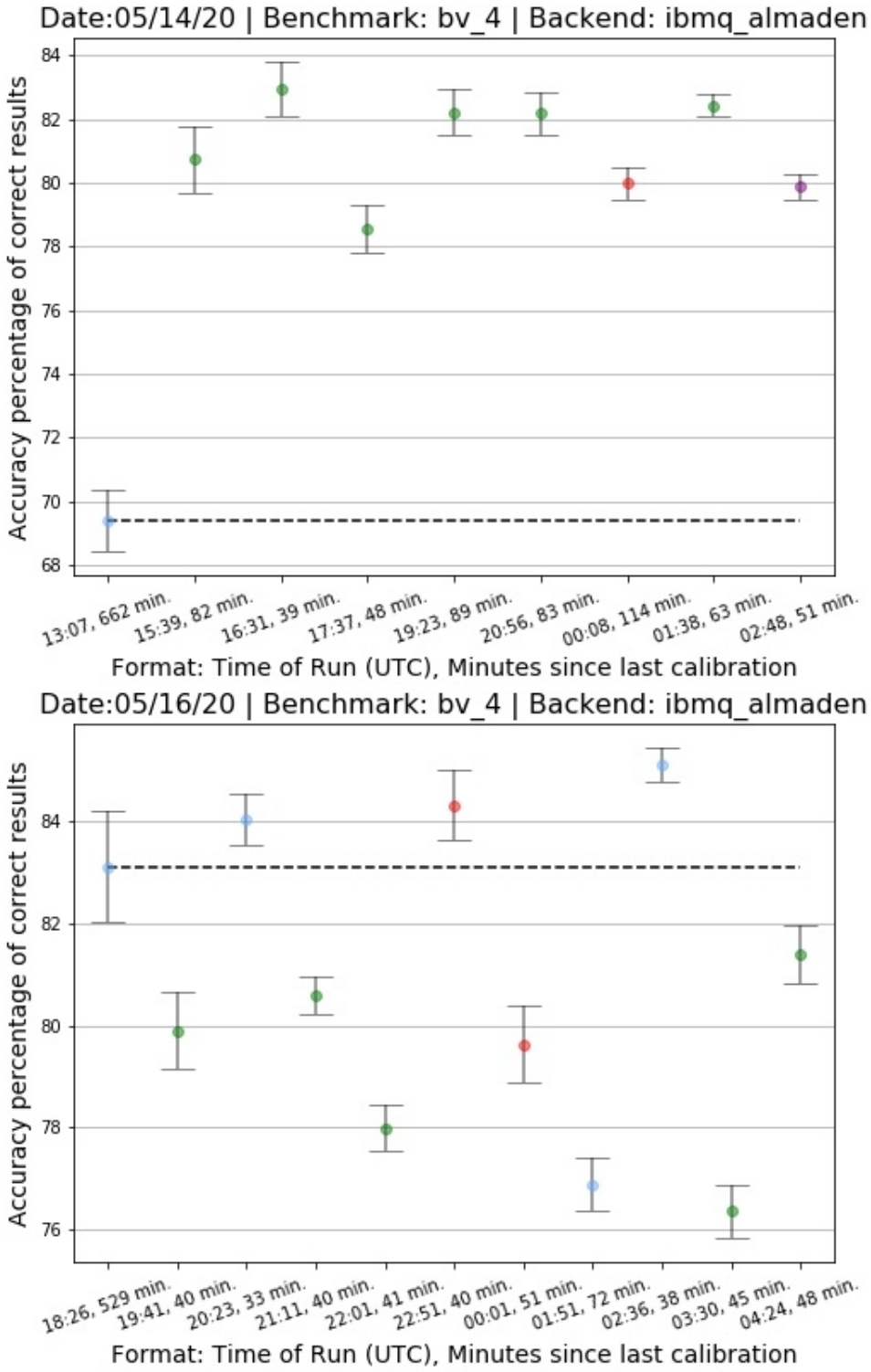


Figure 2.5: *Accuracy of Bernstein-Vazirani with 4 Qubits on 5/14/20 (upper) and 5/16/20 (lower)*

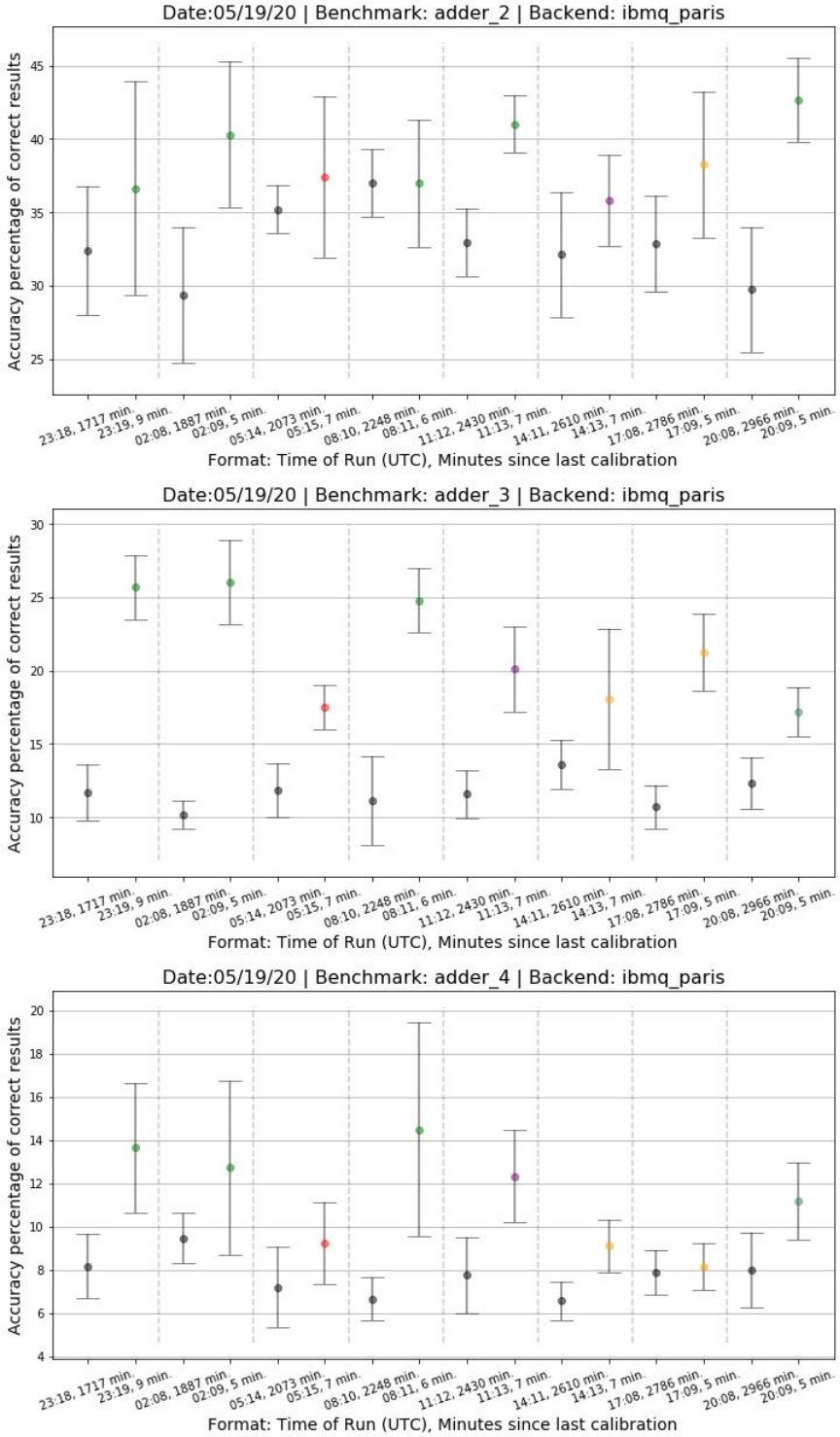
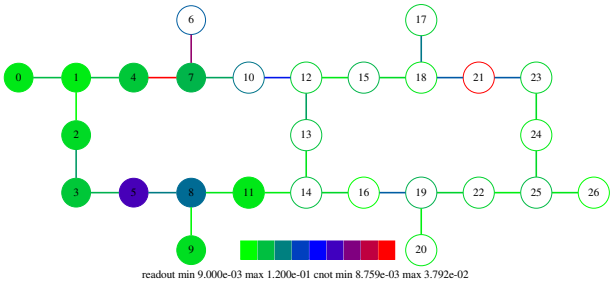
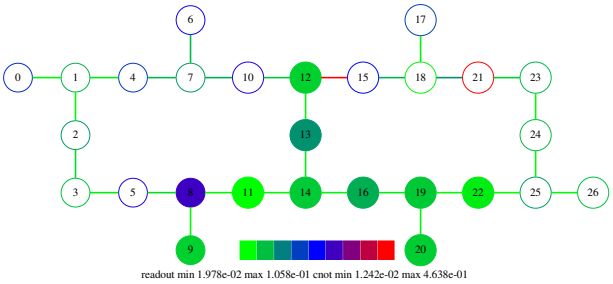


Figure 2.6: *Accuracy of Adder for 4/6/8 qubits (upper/ middle/ lower graphs) at different times during the day with prior calibration in minutes.*

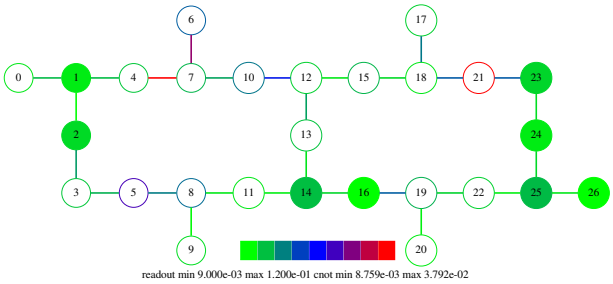
Figure 2.7: *Percent improvement of accuracy for just-in-time transpilation in dedicated mode.*



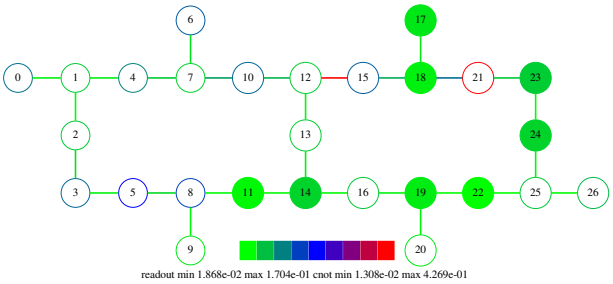
(a) *IBM layout for adder(4) on Paris*



(b) *Our layout for adder(4) on Paris*



(c) *IBM layout for hs(8) on Paris*



(d) *Our layout for hs(8) on Paris*

Figure 2.8: *Circuit Layouts*

CHAPTER

3

EMPIRICAL EVALUATION OF CIRCUIT APPROXIMATIONS ON NOISY QUANTUM DEVICES

3.1 Introduction

Contemporary quantum computing devices are commonly referred to as Noisy Intermediate-Scale Quantum (NISQ) computers as they are fraught by a multitude of device, systemic, and environmental sources of noise that adversely affect results of computations [Preskill, 2018]. A number of factors contribute to noise, or errors, experienced during the execution of a quantum program. These include

- noise related to limits on qubit excitation time and program runtime due to decoherence;
- noise related to operations, i.e., gates performing transformations on the states of one or more qubits;
- noise related to interference from (crosstalk with) other qubits; and

- noise subject to the process of measuring the state of a qubit via a detector when producing a program's output.

Efforts to reduce — or otherwise mitigate — noise are at the front of the effort to create better, more practical quantum computers today [Murali et al., 2019b, Murali et al., 2019a, Wilson et al., 2020, JavadiAbhari et al., 2014, Preskill, 2018, Bishop and Gambetta, 2019, McKay et al., 2018, Zulehner et al., 2018, Tannu and Qureshi, 2019a, Tannu and Qureshi, 2019b, Tannu and Qureshi, 2019c, Li et al., 2019, Li et al., 2020, Wille et al., 2016, Siraichi et al., 2018]. Our work builds upon and complements these previous efforts.

All of these sources of noise have a common characteristic in that noise becomes worse with circuit depth, i.e., the more sequential gates a quantum circuit has, as quantum states (particularly excited states) decohere over time. Today's NISQ devices feature qubits with relatively short coherence times — the longer an excited state has to be maintained, the more noise is introduced, to the point where eventually noise dominates and the original state becomes unrecoverable. Current devices also suffer from noisy or imprecise gates, which add imprecision to a circuit each time a gate is applied, i.e., qubit state diverges slightly from the expected state with the application of each transformation (rotation). Depending on the type of gate, noise varies significantly: Gates operating on two qubits are an order of magnitude more noisy than single qubit gates. Two qubit gates also experience more cross-talk, due to interference with other qubits in close vicinity. Finally, measurement of state (read-outs) is also subject to considerable noise, on par with cross talk and two qubit gates, as opposed to said single qubit noise.

A quantum program expressed as a circuit of gates operating on virtual qubits needs to be translated into a sequence of pulses directed at physical qubits. This translation step (a.k.a. transpilation) offers optimization opportunities to reduce noise. Besides translation of pulses, quantum compilers consider secondary, noise-related objectives to generate optimized quantum programs, e.g., by mapping virtual qubits to less noisy physical qubits (in terms of read-outs) [Murali et al., 2019a, Tannu and Qureshi, 2019c, Tannu and Qureshi, 2019a] and their connections (for two qubit gates) [Murali et al., 2019a, Tannu and Qureshi, 2019b], or even by increasing the distance to reduce cross-talk between qubits for a given device layout [Bishop and Gambetta, 2019, Murali et al., 2020, Ding et al., 2020].

Another angle to address noise is to reduce the depth of circuits. By reducing the number of gates in a circuit, especially the number of two-qubit gates, the depth of the circuit, i.e., the span of time during which qubits remain in excited states, is shortened, which lowers the effect of decoherence. In fact, this may well bring long circuits within reach of short decoherence times that otherwise could not finish on a NISQ device before losing their states. One promising way to reduce the number of gates is to create an *approximate circuit* [Amy et al., 2013, De Vos

and De Baerdemacker, 2015, Alam et al., 2020], i.e., a circuit which does not provide an exact (theoretically perfect) transformation for a target unitary but rather a "close" fit for the unitary. (One could make a comparison to fixed precision arithmetic in classical computation here, which often relies on converging calculations as an approximation of exact numerical results.) On a NISQ device, an exact quantum circuit is prone to develop large error with increasing circuit depth. In contrast, a near-equivalent approximate quantum circuit with shorter depth, even though subject to a slightly incorrect transformation, may have the potential to yield a result that is *closer* to the noise-free (theoretically) desired output. This opens up an interesting trade-off between longer-depth theoretical precision with more noise vs. shorter-depth approximation with less noise. It is this trade-off this work aims to assess and quantify.

The task of finding an approximate circuit is similar to the process of circuit synthesis [Davis et al., 2019, Younis et al., 2020]. Circuit synthesis is another avenue that attempts to reduce circuit depth. Synthesis here refers to the process of what can be considered design space exploration: Given a quantum program, expressed as an exact circuit or an equivalent unitary matrix, other circuits are systematically constructed and then evaluated in a search for an equivalent, shorter depth quantum circuit with the *same* unitary. If found, such a target circuit can be transpiled to a specific machine layout and set of gates with shorter execution time, which may be within the given decoherence threshold of a NISQ device.

The main difference between circuit synthesis and searching for an approximate circuit is that instead of searching for an equivalent (functionally indistinguishable) circuit, the latter searches for an *approximate* circuit of a shorter depth with a *slightly different unitary*. While this leads to inferior results on a noise-free machine, the intuition is that due to noisy gates, shorter, approximate circuits have the potential to outperform longer, more precise circuits.

There are many different metrics which can be used to determine whether two circuits are equivalent. Quantum synthesis compilers [Davis et al., 2019, Younis et al., 2020] typically use distance metrics between "process" representations of the program, such as the Hilbert-Schmidt (HS) distance between the associated unitary matrices, or the diamond norm [Gilchrist et al., 2005, Aharonov et al., 1997]. In the process view, two programs are deemed equivalent when at distance "zero".

In the context of this work aiming at approximation, synthesis is used to find a circuit exceeding a distance of zero relative to the original program so that, when run on a NISQ machine, its output is expected to be close to that of the original program. One challenge with using approximate circuits is that of finding a suitable metric to assess the appropriateness of a set of approximate circuits. One potential option is a process distance, such as HS, within a certain range (threshold). Another is to instead consider output-related metrics, such as the Jensen-Shannon Divergence or Total Variation Distance [Endres and Schindelin, 2003]. This

remains an open question.

The novelty of this work is in its focus on the analysis of a particular use-case of approximate circuits, namely by considering a set of approximate circuits created by quantum synthesis software. When these offered an unworkable number of circuits, we constrained which ones we used by a given HS distance as a threshold. We never choose an HS threshold of less than 0.1, which still results in a wide range of approximate circuits. With a large selection of circuits we can investigate the behavior of *many* approximate circuits in the presence of *different noise levels*.

With this work, we make the following novel contributions to the broader aim of searching for approximate circuits:

- We demonstrate how to obtain a wide range of approximate circuits from custom modified circuit synthesis tools.
- We provide a proof-of-concept that approximate circuits can outperform exact circuits on NISQ devices for small, well known algorithms — Grover's Algorithm and the Multiple-control Toffoli gate — as well as a specific physics application, namely the three-four qubit Transverse Field Ising Model (TFIM).
- We show how the results of approximate circuits change relative to the noise induced by two-qubit errors. Specifically, we assess the effect of two-qubit errors of lower-depth circuits with different approximation thresholds vs. that of the exact, longer circuit. Experiments indicate improvements in overall precision for shorter approximate circuits over longer precise ones by up to 60%.

3.2 Problem Statement and Objectives

This work seeks to assess if approximate circuits can outperform exact circuits on today's NISQ devices. Utilizing approximate circuits ultimately comes with challenges posing four fundamental questions:

1. How can approximate circuits be generated?
2. Can the search for or generation process of approximate circuits be constrained and, if so, how?
3. Will the resulting approximate circuits outperform their equivalent original ones?
4. Can algorithms be designed to make circuit synthesis and the search of resulting circuits scalable?

Before investigating these problems, however, a more fundamental question should be asked: *Is there any value in approximate circuits to begin with?* In other words, can any approximate circuit actually outperform the original circuit at all? It is *this* line of reasoning that our work is trying to answer — before we can explore the more general challenges posed by the four questions above.

In this work, we show that there *is* potential value in approximate circuits: They can outperform theoretically perfect circuits on today's NISQ hardware. We also confirm that any method of selecting appropriate approximate circuits will need to take the noise/error levels of target devices into account.

3.3 Design

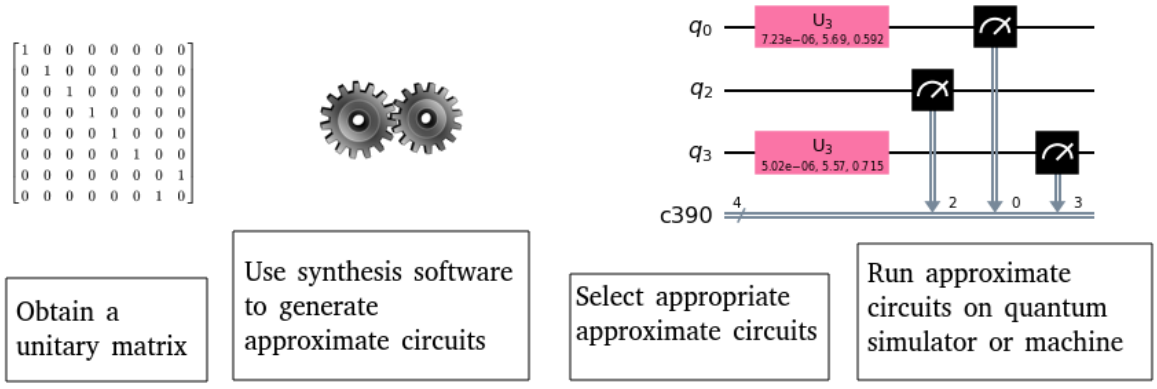


Figure 3.1: Generic workflow of using approximate circuits. The example is an approximation of the first timestep of the TFIM circuit.

One way to find approximate circuits is to look at approximate circuits generated by the intermediate steps of circuit synthesis programs. These programs do not typically scale to a level which would make them an ideal way to create large approximate circuits in practice, but are an easy way to create them as a proof of concept. Synthesis programs typically look for the shortest circuit they can find with a distance of some kind at "zero". As these programs are interested in finding as short a circuit as possible, they tend to investigate many shorter circuits before finding their target; these circuits are already nearly optimized for their layout, making them ideal approximate candidates.

Before utilizing synthesis software to generate approximate circuits, we typically need to alter the synthesis tools to produce as output, besides a single circuit, additional circuits that

are farther away from the target. These tools already generate and test many circuits during the search for an equivalent circuit, which allows our enhancements to integrate naturally with the existing flow within synthesis tools.

Figure 3.1 shows the workflow of our process. We first need to obtain our target unitary. Quantum operations can be represented by matrices, and the target unitary is the result of multiplying these transforming matrices of a circuit (or subcircuit) that is to be approximated. In IBM's Qiskit python interface [Aleksandrowicz et al., 2019], the unitary of a circuit can be obtained with the following command on the target QuantumCircuit object $circuit$:

$$matrix = qiskit.quantum\_info.Operator(circuit).data$$

The second step is to use our altered synthesis software to generate approximate circuits for our target matrix.

Third, given the the enhanced synthesis software that outputs every circuit it checks, we need to select which approximate circuits we want to check. How to perform this selection is still an open question. For our analysis, we intend to compare a large number of circuits, so we select many circuits with little to no filter for pre-selecting the most accurate ones.

Finally, the selected circuits need to be run on a quantum machine or simulator. For our study, we then compare the results with the expected output, either a known value or our original circuit run on a simulator with no errors.

3.4 Implementation

For our exploration of approximate circuits, we use two different synthesis tools, QSearch and QFast, both of which are part of the Berkeley Quantum Synthesis Toolkit (BQSkit).

The QSearch [Davis et al., 2019] optimal depth circuit synthesis tool builds a sequence of circuits of increasing length and decreasing HS distance until it finds the first circuit with a distance of "zero", a value which can be specified but which defaults to less than 1e-10. It does this by following the A* algorithm. Specifically, it explores different branches of the circuit space by adding on blocks of three gates. Certain machine layouts can be taken into account by restricting these blocks to only being placed between connected qubits. These blocks are made up of one two qubit controlled NOT (CNOT) gate and two single qubit U3 gates on each of the same qubits. The U3 parameters are optimized using one of a number of different numerical optimizers, including COBYLA and BFGS, provided by SciPy 1.20, and reoptimized after each step. This optimization ensures that, for this specific layout of CNOT and U3 gates, this circuit is the closest possible to the target. Because it considers each option, this is guaranteed to be

depth optimal with respect to two-qubit gates.

The QFast [Younis et al., 2020] synthesis tool likewise builds a sequence of circuits of increasing length, but it has a more complicated algorithm for finding circuits of increasingly higher quality. QFast is not guaranteed to be optimal and gives less of a choice of approximate circuits, but handles circuits with more qubits than QSearch within acceptable search times.

In our work we enhance the QSearch software such that instead of saving only the final circuit, it also saves every intermediate circuit during its search. We then select a portion of the circuits, always with a maximum HS distance threshold of at least 0.1, in order to have a wide range of circuits but none which differ entirely from the target circuit. QFast requires no source code alteration, but it needs to be given a dictionary with the key of "partial_solution_callback" pointing to a function to output these solutions. This dictionary is used by calling QFast with the keyword "model_options".

These circuits are then executed in three different methods. First, they are executed on the IBM Qiskit [Wood,] simulator using hardware specific (ibmq_ourense, ibmq_toronto, ibmq_-manhattan, ibmq_rome, ibmq_santiago) noise models. These noise models are created using error data collected from IBM's own physical machines, creating a noisy simulator.

Second, they are executed on noise level sweeps. These use the ibmq_ourense noise model as a base, but change the two-qubit gate noise level during a sensitivity study in order to observe the effect of different types and levels of noise.

Finally, The approximate circuits are also executed on the ibmq_ manhattan, ibmq_toronto, and ibmq_rome physical machines.

3.5 Experimental Framework

We selected three different algorithms for the evaluation. We start with circuits generated [Bassman et al., 2020, Bassman et al., 2021] for the time-dependent Transverse Field Ising Model (TFIM). The TFIM is a quintessential model for studying various condensed matter systems, and its time-dependent manifestation shows promise for revealing new information about non-equilibrium effects in materials. Current algorithms for designing quantum circuits for the simulation of such models, however, produce circuits that increase in depth with the growing number of time-steps; circuits quickly grow beyond the NISQ fidelity budget, placing tight limits on the number of time-steps that can be simulated. This class of circuits, therefore, stands to greatly benefit from shorter, approximate circuits. In addition, the output for these circuits can be condensed to a single number to easily be compared to the output of the approximate circuits, allowing for an easy target for the approximate circuits.

We next study Grover's algorithm [Grover, 1996] followed by the multi-control Toffoli

gate [Toffoli, 1980] to demonstrate the general capability of our method.

We decided to focus on small circuits for this work due to NISQ and synthesis limitations. We use the three and four qubit execution of the circuits for most of our experiments, and scale up to five qubits with the multi-control Toffoli gate. For TFIM, we assess at the first 21 time steps of 3ns. This results in 21 different circuits for different times in the evolution of the magnetization. All of these circuits are related, but they can also be investigated individually.

Table 3.1: Average CNOT errors on a selection of IBM physical machines as of 2021/01/18

IBM Machine	Num. qubits	Av. CNOT err.
Manhattan	65	.01578
Toronto	27	.01377
Santiago	5	.01131
Rome	5	.02965
Ourense	5	.00767

Table 3.1 provides a snapshot of typical CNOT error rates at the time of writing. they give a contemporary view of the types of CNOT errors that we compare against and reflect the constant changes of NISQ devices with different error rates on different qubit connections even on the same device.

For our experiments using simulators we transpile under IBM's optimization level 1 with mappings to qubits 0, 1, 2, 3, and 4. Our experiments on physical machines are transpiled under optimization level 3, which at the time of writing allows IBM to map virtual qubits to the best available physical qubits. All work is performed with Python 3.8.2 and Qiskit 0.18.3, Qiskit-aer 0.5.1, Qiskit-ibmq-provider 0.6.1, and Qiskit-terra 0.13.0. Our QSearch enhancements are based on search_compiler version 1.2.1, and we used QFast version 2.1.0.

3.6 Results

We first report experimental results for simulations under given noise models of contemporary quantum devices subject to NISQ constraints. We then perform a sensitivity study on the effect of noise levels, including both smaller (future) and larger (past) noise levels than seen on the reference device, still using simulation. This is followed by experiments on IBM Q devices with approximate circuits under default transpilation with full optimization. Finally, we perform a sensitivity study investigating the effect of how approximate circuits are mapped to qubits on hardware devices with respect to noise level, particularly of CNOT gates.

3.6.1 Noise Model Simulations

We first investigate the noise and approximation quality of our approach. Figure 3.2 depicts results for a 3-qubit TFIM problem under the Toronto (IBM Q) noise model with magnetization (y-axis) over time steps (x-axis) in 21 intervals of 3ns each. Series “Noise free reference” shows the result for the circuit generated by the TFIM domain generator and simulated on the ideal hardware. This is the target for the other circuits; the closer they are to these results, the better they are. Series “Noisy reference” shows the behavior of the same circuits when simulated with the hardware specific noise model. “Noisy reference” behavior quickly diverges from the ideal as circuits become more complex with increasing timesteps. Series “Minimal HS” shows the behavior of the synthesized circuits when using process metrics (HS) as the quality indicator. As these are much shorter (six CNOTs versus tens of CNOTs for the reference circuit) than the baseline implementation, their results are typically closer to the ideal results.

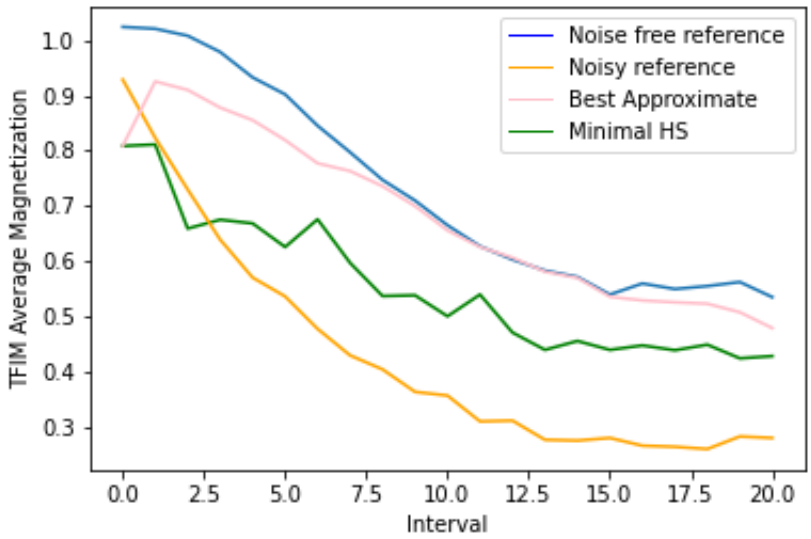


Figure 3.2: Magnetization over 21 timesteps of selected (best/minimal HS) approximate circuits for the 3-qubit TFIM using the Toronto error model.

The potential of approximate circuits is depicted by Series “Best approximate”, where we select the circuits with “best” output behavior. Their CNOT depth is always shorter than the HS=0 circuits, and so even though the process distance is greater they provide a result closer to the noise free reference. This was also observed across other noise models.

Observation 1: Short approximate circuits can outperform long circuits with a lower process distance in simulation under device noise models.

Let us investigate the range of solutions generated by approximate circuits in more detail. Figure 3.3 shares the noise free and noisy reference data series with Figure 3.2, but it additionally includes dots representing each approximate circuit. The colors of the dots indicate how many CNOTs were used in the approximate circuits; in this case, red dots represent two CNOTs and blue dots represent six. It can be seen that while there was a wide difference in accuracy over the different approximate circuits, nearly all of them performed better than the noisy reference.

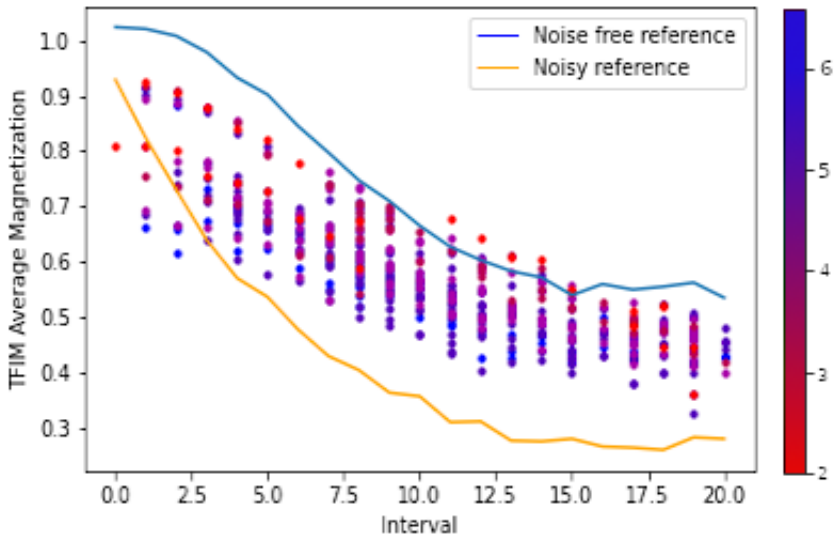


Figure 3.3: Magnetization over 21 timesteps of all approximate circuits for the 3-qubit TFIM using the Toronto error model.

We next investigate the impact of circuit width (in qubits) and depth (in CNOT gates) for the same TFIM application. Figure 3.4 represents the four qubit TFIM circuit with the same line graphs again. The number of CNOTs in an individual circuit in this case can range from 1 to 48, which illustrates the wide range of approximate circuits, many of which are closer to the noiseless reference than the noisy reference is.

We now turn our investigation to the impact of circuit approximation for different algorithms and circuits, first with Grover and then with Toffoli. Figure 3.5 depicts results for Grover's algorithm with a search target of '111' over eight boxes, where each dot represents a circuit. The blue dots each indicate an approximate circuit, while the orange dot and the line represent the output circuit of the hand-coded reference implementation with nine CNOTs. Figure 3.5 shows the quality of the circuits as the probability of selecting the correct box (y-axis), where higher probability is better. Here, CNOT count is shown on the x-axis rather than indicated by

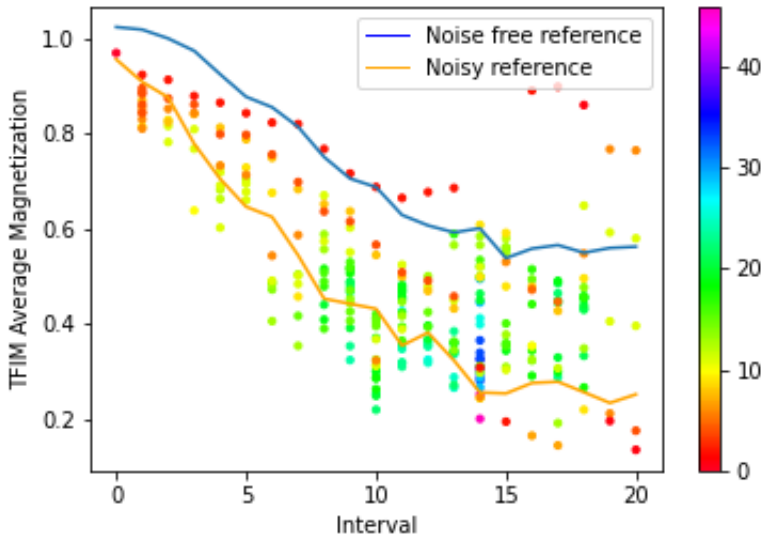


Figure 3.4: Magnetization over 21 timesteps of approximate circuits for 4 qubit TFIM sing the Santiago noise model.

color coding. This shows a wide array of approximate circuits, many of which outperform the reference; only a smaller fraction (below the dashed line) underperform. The challenge here is to select a "good" approximate circuit from the wide array of possible candidates. We observe and investigate this challenge with different metrics but its solution is ultimately beyond the scope of this paper.

Observation 2: To capitalize on the potential of approximate circuits, a selection method and an associate metric are required to ensure superior performance under noise.

We further perform experiments for the Toffoli gate with different numbers of qubits. Figure 3.6 shows the results for four qubit Toffoli gate—that is, three control qubits to one target qubit. We use the Jensen Shannon (JS) distance [Endres and Schindelin, 2003] to analyze these circuits (y-axis), as the Toffoli gate can be programmed to represent a variety of functions, each with different (but known) output. We test each approximate circuits for a subset of such functions and parameters since a given circuit results in different probabilities for correct output. The JS distance provides a composite metric to reflect accuracy (lower is better in this case).

The four qubit results indicate that low-depth approximate circuits outperform those with high CNOT depth. The orange dot on the dashed line represents Qiskit's multiple-control Toffoli gate without any ancilla bits while the red dot indicates QFast's default result of an equivalent circuit. The JS metric indicates that the former (orange) outperforms the latter (red). Furthermore, many deeper approximate circuits perform worse than Qiskit's Toffoli without

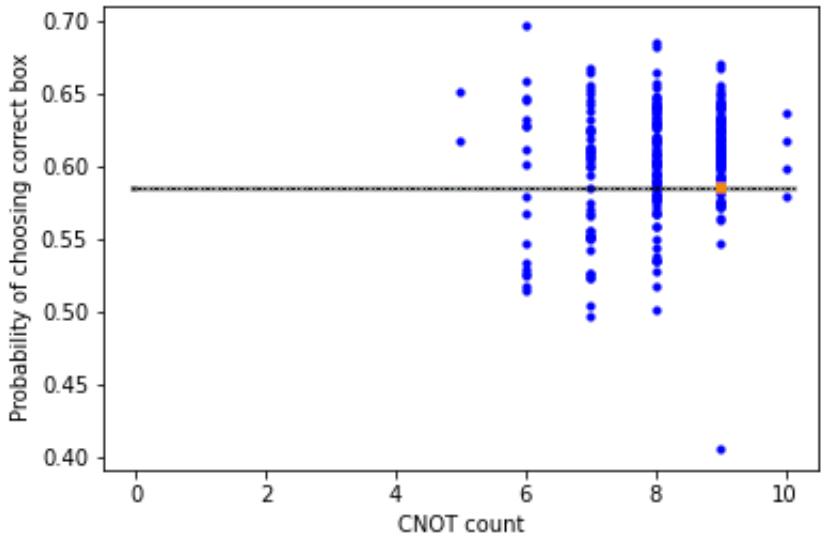


Figure 3.5: Probability of correct result over CNOT count of approximate circuits for 3 qubit Grover's algorithm using the Toronto noise model. Reference circuit in red.

ancilla while shorter approximations (below the line) can provide even better results than Qiskit. This implies that there is room for improvement even over the reference implementation of given circuits on today's noisy machines.

Observation 3: Approximate circuits generated from synthesis can outperform discrete reference circuits under noise.

Figure 3.7 depicts results for a five qubit Toffoli gate, again without any ancilla qubits for either the reference or approximate circuits. These results reinforce the earlier four qubit results: The JS distance of the reference circuit is higher for five qubits, but some approximate circuits have a distance even closer to zero than the best of those for the four qubit Toffoli gate. The correlation of shorter circuits performing better is evident, but outliers exist. As the number of CNOTs increases to the hundreds, the JS values approach 0.465. This is significant because, in this implementation, random noise (an equal number of results of 00000 as 00001 as 00010 and so on) results in a JS distance from the target of 0.465.

We also performed experiments for a 3-qubit Toffoli gate. In this case, the 3-qubit approximate circuits performed poorly compared to the optimized hand-crafted Toffoli gate commonly used, which uses only 6 CNOTs (graph omitted). This illustrates that simple, short circuits provide little benefit for approximations via QSearch or QFast whereas deeper and more complex ones can benefit significantly for today's noisy quantum hardware. It also presents a challenge for synthesis tools as wider circuits (beyond 6-8 qubits) with corresponding depth results in excessive search cost.

Observation 4: The benefit of using approximate circuits increases with the depth of the

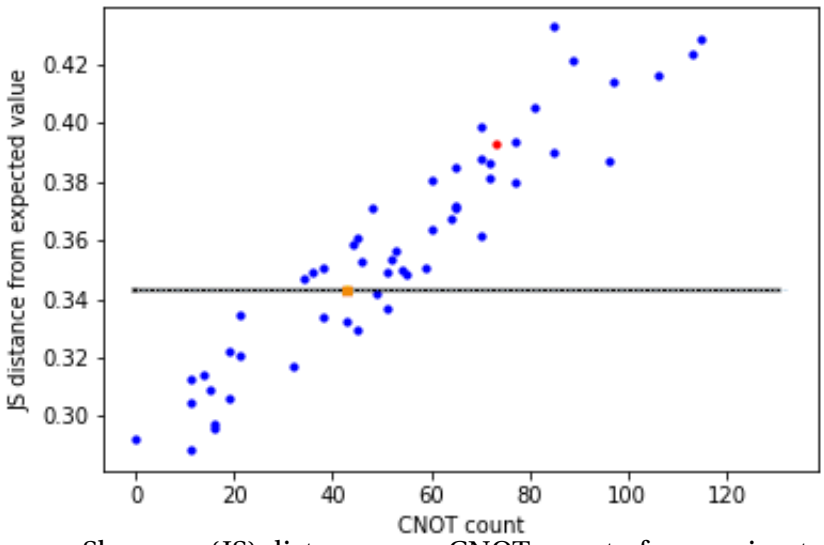


Figure 3.6: Jensen Shannon (JS) distance over CNOT count of approximate circuits for 4 qubit Toffoli compared to the reference circuit using the Manhattan noise model. Qiskit (orange) and QFast (red) circuits are outperformed by other approximate circuits.

reference circuit.

3.6.2 Error Sensitivity Studies

We assess the sensitivity of approximate circuits to noise. To this end, we use the Ourense noise model as a base but change the CNOT error rate to assess how the performance of circuits changes results in response. Figures 3.8, 3.9, and 3.10 present the approximate circuits with increasing noise. Circuits are again color coded using their depth, with red circuits consisting of two CNOTs and blue circuits of six. The lines of these colors represent the best performing approximate circuits for that number of CNOT gates.

Figure 3.8 depicts simulations for a CNOT noise level of zero. It illustrates the spread of circuits with different noise sources (with the exception of CNOT noise), and shows that CNOT depth is not closely correlated to the quality of results with no CNOT noise.

Figure 3.9 shows the simulation for a CNOT error of 0.12, similar to that of today's lowest quality physical devices, and assesses the impact on performance. Note that the increase in CNOT error is accompanied by a decrease in the observed average magnetization. Many of the longer circuits in blue or purple, which were covered up by the red dots, become visible showing that a diverse number of approximate circuits react differently under CNOT noise.

Figure 3.10 depicts simulations for a CNOT error of 0.24, which is worse than many current IBM machines and reinforces this trend. These results are promising. We clearly see that some

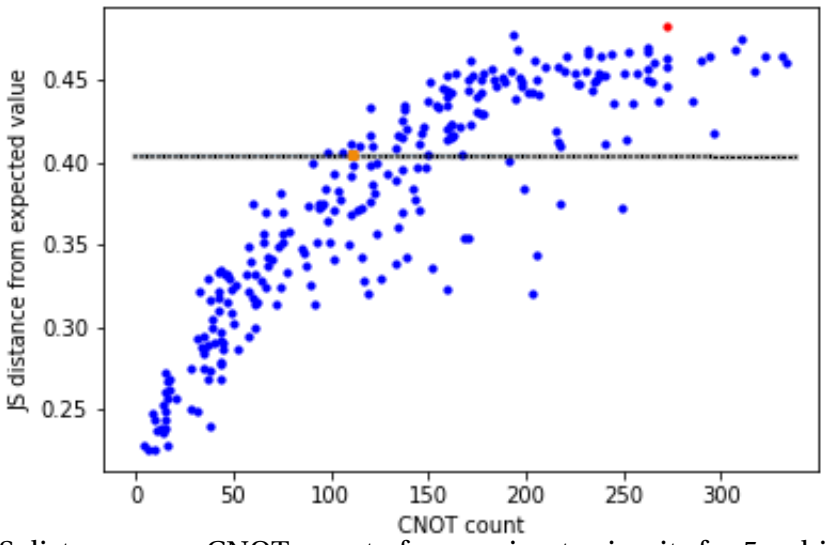


Figure 3.7: JS distance over CNOT count of approximate circuits for 5 qubit Toffoli compared to the reference circuit using the Manhattan noise model.

individual circuits improve, i.e., more closely approximate the error free reference, with an increase in two-qubit error. We also see that deeper circuits are more affected by CNOT error than shallower circuits. With a low two-qubit error, many of the deeper circuits lie on the line corresponding to the error free reference. As this error increases, these deep circuits quickly decline in quality, and the shallower circuits perform relatively better. This is seen as the best of the longest circuits perform worse than the best of the shortest circuits for all timesteps; but without CNOT noise, this is not necessarily true. Some of these circuits actually benefit from the noise and more closely approximate the error free reference.

The takeaway from this trend is that different approximate circuits should be chosen based on the error levels of the physical machine. A program can afford to use a long circuit on machines with low error, but a noisier machine will benefit from a shorter, approximate circuit.

Observation 5: Beyond merely being less affected by noise than the reference circuit, some approximate circuits perform better in the presence of noise. This performance increase is dependent on the noise parameters of the system.

Figure 3.11 further supports this by depicting the depth of the best performing circuit for different noise levels. A trend can be seen: the worse the error (the more red the line), the shallower the circuits with the highest performance in general, but not under all circumstances. A similar trend is seen with our other algorithms (figures omitted).

These results generally support the initial conjecture: as the amount of noise in the models increases, the output quality of deeper circuits deteriorates more quickly than that of the

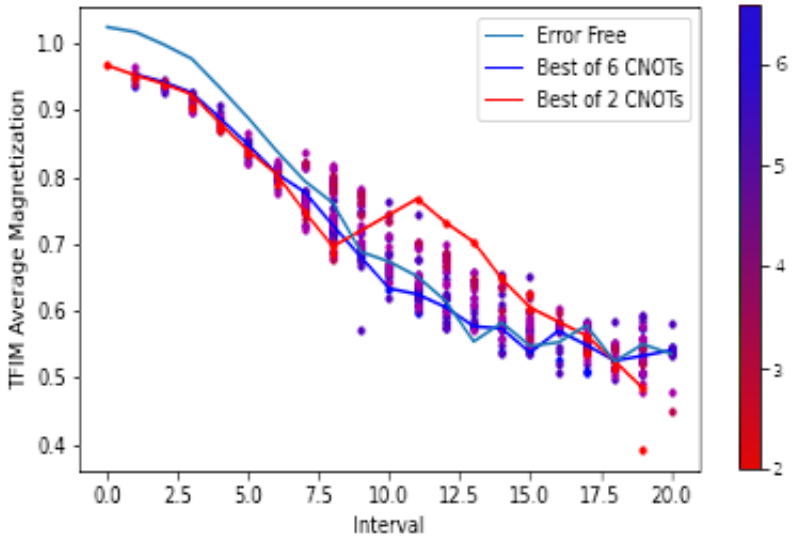


Figure 3.8: Magnetization over 21 timesteps of approximate circuits for 3-qubit TFIM using the Ourense noise model with no CNOT error.

shallower circuits. This causes some of the shallower circuits to produce results that are closer to the ideal results than the deeper circuits, even though the deeper circuits would perform better on an ideal, noise-free machine. This is most noticeable with circuits that contain many CNOTs and on noisier models. It is less noticeable with circuits which are already short or simulated on models of low noise.

Observation 6: The greater the level of two-qubit noise on the target machine, the more benefit is gained from short approximate circuits.

3.6.3 Results on IBM Q Hardware

Figures 3.12 and 3.13 depict results from running the three and four qubit TFIM circuits on contemporary IBM quantum hardware devices. These results provide insight on how much can be gained from using approximate circuits in practice today. We observe that almost all of the approximate circuits in Figure 3.12 and the large majority of the approximate circuits in Figure 3.13 perform better than the default circuits.

We also observe that the approximate circuits here are distributed similarly to Figure 3.9, showing that the earlier constructed noise models are not far off from actual noise on hardware today.

Observation 7: Approximate circuits can perform well compared to reference circuits on real quantum hardware devices as well as on noisy simulators.

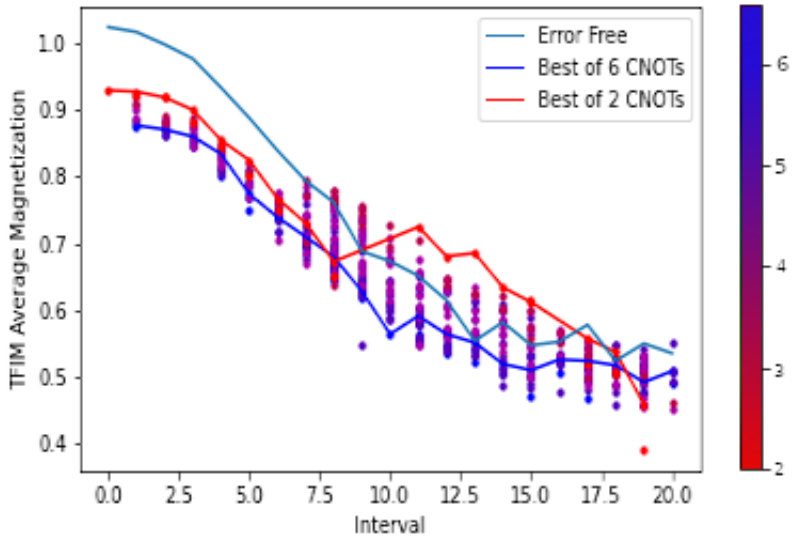


Figure 3.9: Magnetization over 21 timesteps of approximate circuits for 3-qubit TFIM using the Ourense noise model with a simulated CNOT error of 0.12.

Figure 3.14, similar to Figure 3.5, depicts results from experiments with the 3 qubit implementation of Grover's algorithm. As before, many (but not all) of the approximate circuits perform better than the reference circuit. There is a minor bias to shorter circuits performing better, but not a significant one. It should be noted that the reference circuit here had more than 50 CNOTs and is thus omitted from the figure. The line is still at the performance level of the reference circuit.

Figure 3.15 shows the result of the 4 qubit Toffoli and its approximates on a real machine. At first, the result looks similar to the distribution in Figure 3.6. However, while the best approximate circuits do have a much lower JS score (by 78%) than the reference circuit (orange), the reference circuit and many of the approximate circuits actually perform worse than random noise (as mentioned in the context of discussing Figure 3.7, random noise has a distance of 0.465).

This indicates that even the approximate circuits are still too noisy to run on the physical machines, but we expect them to perform better than the reference circuit when run on less noisy devices.

Observation 8: Trends indicate a continuing potential of approximate circuits to outperform reference circuits in the near future, even as noise levels in physical machines decline.

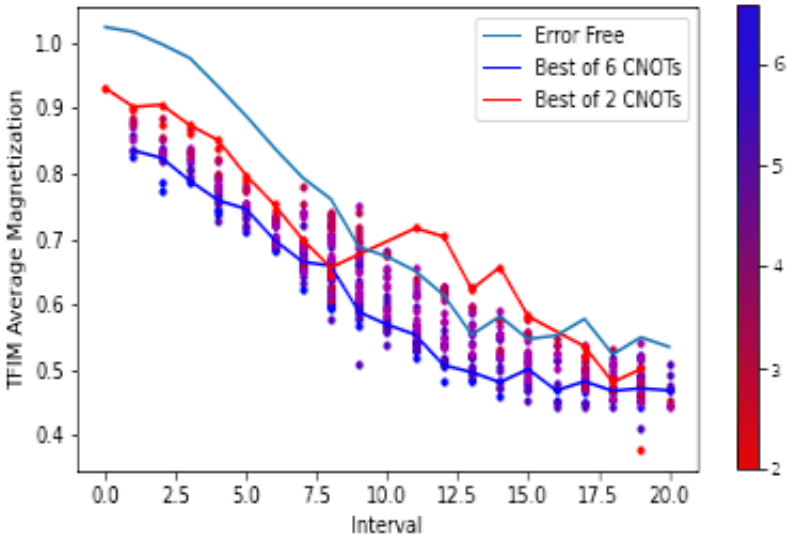


Figure 3.10: Magnetization over 21 timesteps of approximate circuits for 3-qubit TFIM using the Ourense noise model with a simulated CNOT error of 0.24.

3.6.4 Sensitivity to Qubit Mappings on IBM Q Hardware

We further investigate the impact of mapping circuits to specific qubits with CNOT resonance channels of different noise levels for the IBM Toronto physical quantum device using the 4 qubit Toffoli. The qubit connectivity of this machine is shown in Figure 3.16, as reported by IBM on the day of experimentation. The nodes represent different qubits, and their color indicates the readout error in the range depicted on the upper heatmap index to the left. The edges represent the connection between the qubits, and their color indicates the CNOT error level on the lower heatmap index.

Experiments are conducted with four different (manual) mappings for the approximate circuits plus one (automatic) mapping using Qiskit's transpiler at optimization level 3. We depict only the circuits with the best and worst results here.

Figure 3.17 shows results for circuits mapped onto the qubits within the blue circle in Figure 3.16. These results exhibit the shortest JS distance of ≈0.4 (best), and about a third of the circuits lie below the reference of ≈0.47.

Figure 3.18 depicts the results for mappings into the red circle, which provided the worst results with higher JS distance (reference: JS≈ 0.485, approximate circuits start at JS≈0.45) than that of any other mapping. Other mappings (not depicted) lie in between these results.

Figure 3.19 shows the results of transpiling the same approximate circuits with Qiskit under level three optimizations. As each approximate circuit was mapped individually and automatically by Qiskit, no single mapping can be reported. The green circle shows the mapping

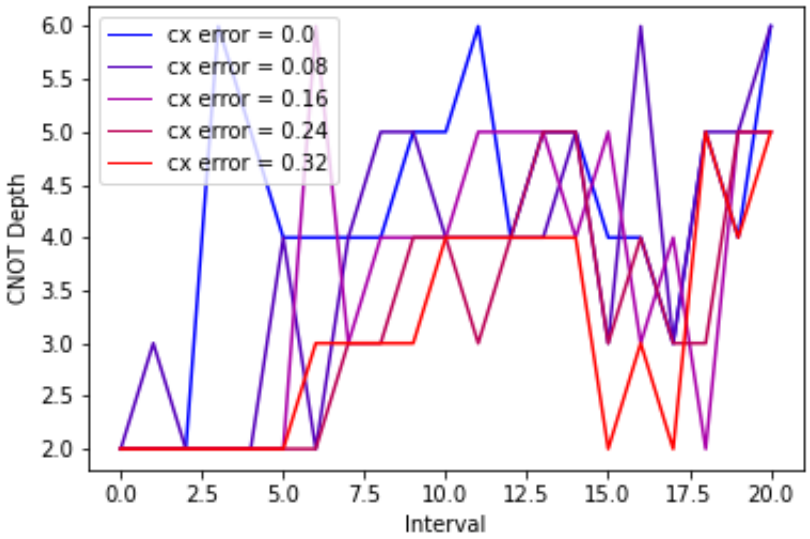


Figure 3.11: CNOT depth over 21 timesteps of approximate circuits for TFIM showing the best approximate circuits for select CNOT errors.

for the best performing circuit within that run, and yellow indicates that of the reference circuit. Fewer circuits have a lower JS than the reference ($\approx .0.46$) but they start as JS$\approx$0.42.

These results are interesting when considering the noise levels in Figure 3.16. The yellow reference circuit (with results in Figure 3.19) chooses two connections with relatively high noise and utilizes about 40 CNOTs, but qubits have relatively high readout fidelity. Nonetheless, it performs better than than the reference circuit in Figure 3.18, which has relatively good connections and only 30 CNOTs. The results indicate that CNOT error cannot be the only source of noise influencing results.

Likewise, the blue mapping has one bad connection, but it provides the best performing circuits (Figure 3.17) with about the same readout fidelity as for yellow. The worst results (Figure 3.18) contribute few (if any) good circuits, yet benefit from relatively good connections but lower readout fidelity according to IBM's noise data.

We know from Observation 6 that increasing CNOT error provides additional opportunities for approximate circuits. Our mapping study is an indication that other noise sources contribute as well, particularly read-out errors (as depicted in Figure 3.16) as well as cross-talk (not reported by IBM but also known to be of the same magnitude). This aspect requires further investigation.

Observation 9: Sources other than CNOT error appear to contribute to the performance of approximate circuits.

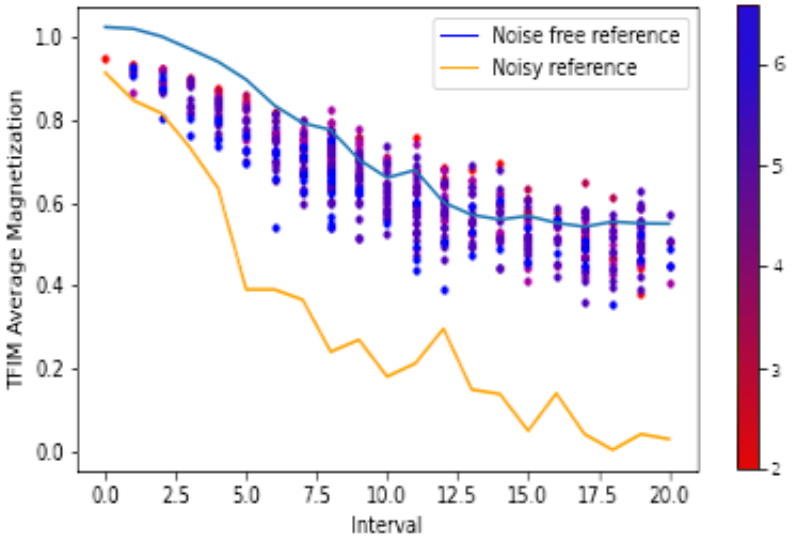


Figure 3.12: Magnetization over 21 intervals of approximate circuits for 3 qubit TFIM on the Manhattan physical machine.

3.6.5 Roadmap and Future Work

Noisy gates enable and encourage circuit approximations. We plan to extend this study and correlate circuit behavior with commonly accepted hardware evaluation metrics, such as gate, read-out, and cross-talk fidelity, and also "quantum volume" [Bishop et al., 2017]. This will allow us to project the potential of approximations in the face of continuous hardware evolution and decreasing noise. Metrics such as quantum volume capture the impact of the relatively short chip coherence times. A "small" quantum volume indicates there are empirical practical bounds on the circuit depth, where we can expect approximations to benefit. Finally, best circuit selection is performed using simulation/execution and examining the result in its specific context. In order to guide circuit generation and synthesis from first principles, we are interested in a thorough analysis of the numerical value of different metrics (Hilbert-Schmidt distance [Gilchrist et al., 2005], Kullback-Leibler divergence [Kullback and Leibler, 1951], Jensen-Shannon distance [Endres and Schindelin, 2003], etc.)

We are also looking into both deeper and wider circuits. QSearch begins to require a prohibitive amount of search time when exposing it to more than four qubits. QFast is a little faster and can typically work with up to six qubits, but is still restricted in the number of qubits it can handle. The Berkeley Quantum Synthesis Kit recently acquired another method of synthesis, QFactor [Younis,], with the ability to synthesize circuits of up to eight qubits. QFactor may be able to create approximate circuits in that range, but a new method of developing approximate circuits is needed for even wider circuits.

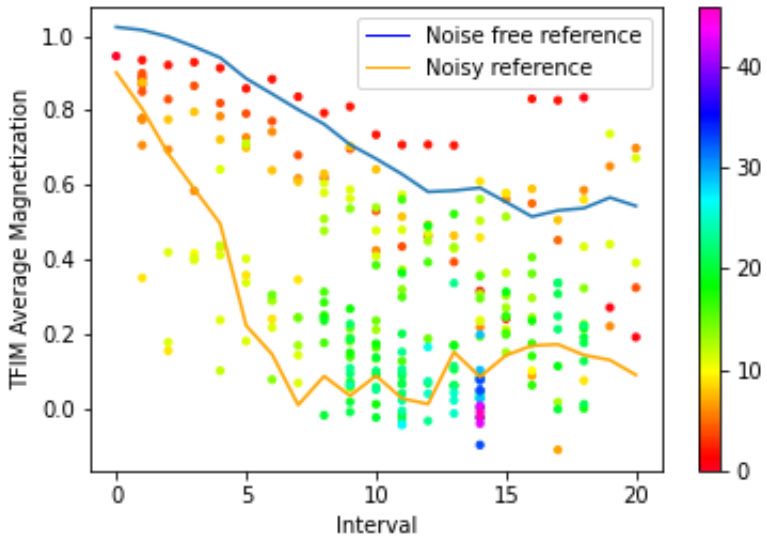


Figure 3.13: Magnetization over 21 timesteps of approximate circuits for 4 qubit TFIM on the Manhattan physical machine.

One possible solution to consider is that of breaking a large program into pieces; it may be possible to create a large circuit out of many small circuits, and we are interested in assessing if approximate circuits also prove to be useful in such a context.

3.7 Related Work

While finding an approximate circuit is typically seen as less desirable than finding an exact circuit, much work has been put in in an effort to finding approximate circuits. The Solovay-Kitaev algorithm [Dawson and Nielsen, 2005] is well known to generate quantum gates which have a specified accuracy.

Work on circuit synthesis [Davis et al., 2019, Younis et al., 2020, Amy et al., 2013, De Vos and De Baerdemacker, 2015] is often classified as either “exact” or “approximate”. But even the approximate algorithms often end up finding closer approximations for circuits than those we are interested in; the small allowable error does not add enough wiggle room to take advantage of short circuits. We are most interested in ϵ-approximate synthesis techniques, which can be coarsened to find circuits which are “more approximate”. Closely related is the Quantum Fast Circuit Optimizer (QFactor) [Younis,], a newly developed piece of synthesis software being distributed as part of the Berkeley Quantum Synthesis Toolkit, just as QSearch and QFast are. It can handle a greater number of qubits than QSearch and QFast can, but is focused more on circuit optimization than just synthesis, and works through tensor networks.

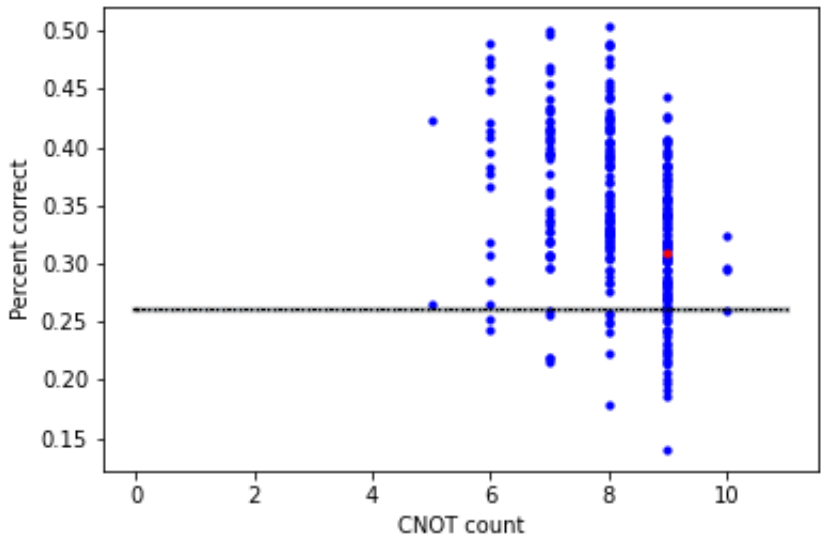


Figure 3.14: Probability of correct results over CNOT count of approximate circuits for 3 qubit Grover's Algorithm on the Rome physical machine.

The Quantum Approximate Optimization Algorithm (QAOA) [Farhi et al., 2015] can also be said to create approximate circuits, though it differs from the work done here in that there is not a known target circuit. Much work has gone into optimizing QAOA circuits. Especially interesting with relation to this work is the work of [Alam et al., 2020], which reorders gates in order to reduce circuit length, similarly finding that approximate circuits with fewer CNOTs tend to outperform approximate circuits with more.

Much more work is being done on other ways to reduce noise or circuit depth [Ding et al., 2020, Murali et al., 2019b, Li et al., 2019, Wilson et al., 2020, Gokhale et al., 2020, Bishop and Gambetta, 2019, Wille et al., 2016]. We are optimistic about these being able to work alongside approximate circuits, though it is unclear whether the benefits of approximate circuits will hold for process which require post-processing or manipulation of error levels, as these may end up interfering with the noise which the approximate circuits rely on to perform better than exact circuits.

3.8 Conclusion

Experimental results confirm that on NISQ devices approximate circuits have the potential to outperform theoretically precise reference circuits. Even though these circuits would perform worse on a perfect machine, if they are created to be similar to the reference circuits but have fewer CNOT gates, these approximate circuits produce higher fidelity results.

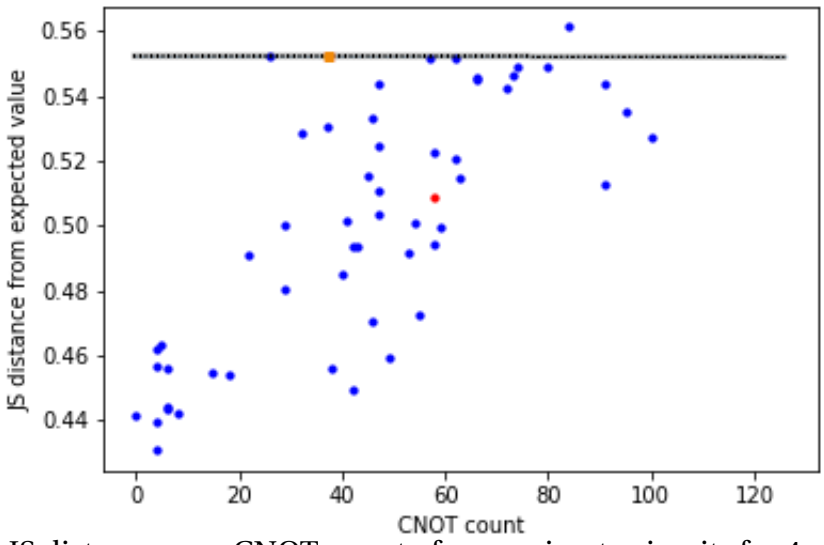


Figure 3.15: JS distance over CNOT count of approximate circuits for 4 qubit Toffoli on the Manhattan physical machine.

Because these improvements rely on reducing the number of CNOT gates, we see approximate circuits perform best relative to reference circuits in situations where the reference circuit has many CNOT gates, namely by up to 60% in experiments.

We have shown that approximate circuits can show greatly increased performance, but we have also shown that selecting the proper approximate circuit is more complicated than comparing process metrics. At the very least, target machine noise levels need to be taken into account. Finding a reliable way to determine the ideal approximate circuit remains an open problem.

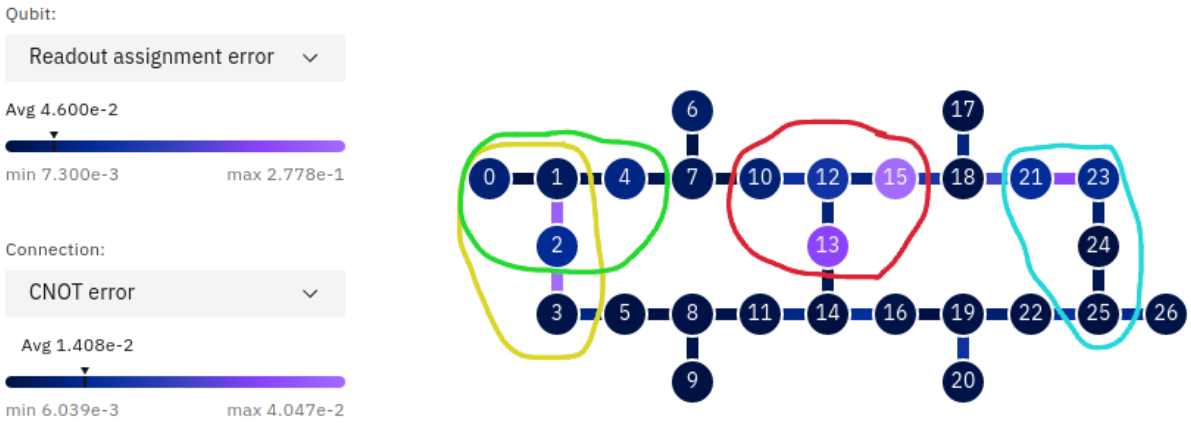


Figure 3.16: Noise report from IBM for their Toronto machine at the time of study. Different circles represent different mappings.

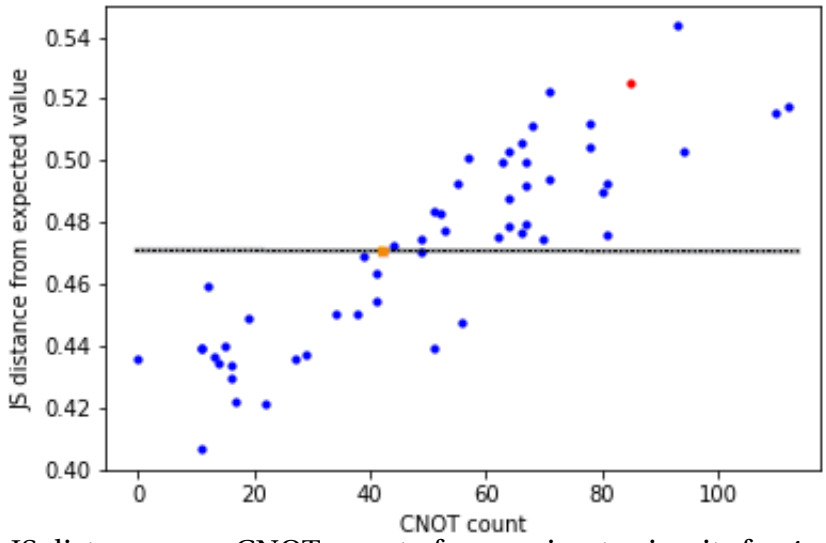


Figure 3.17: JS distance over CNOT count of approximate circuits for 4 qubit Toffoli on the Toronto physical machine showing the best performing mapping.

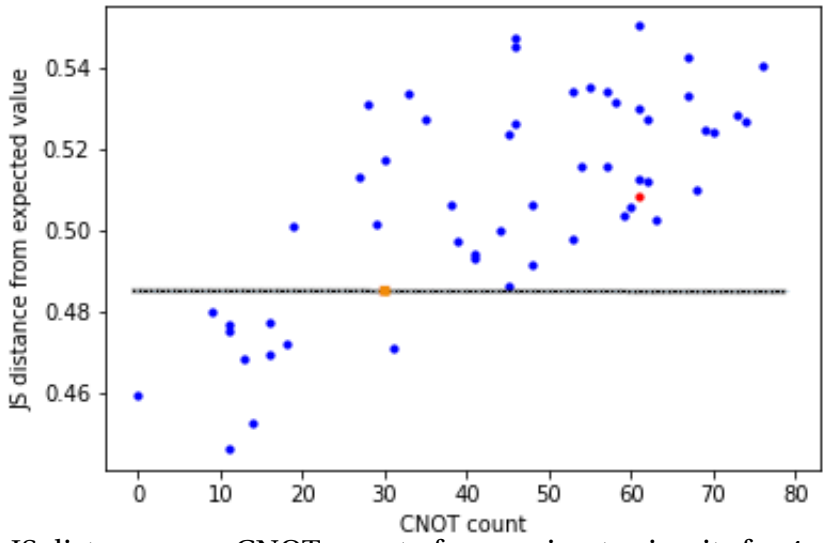


Figure 3.18: JS distance over CNOT count of approximate circuits for 4 qubit Toffoli on the Toronto physical machine showing the worst performing mapping.

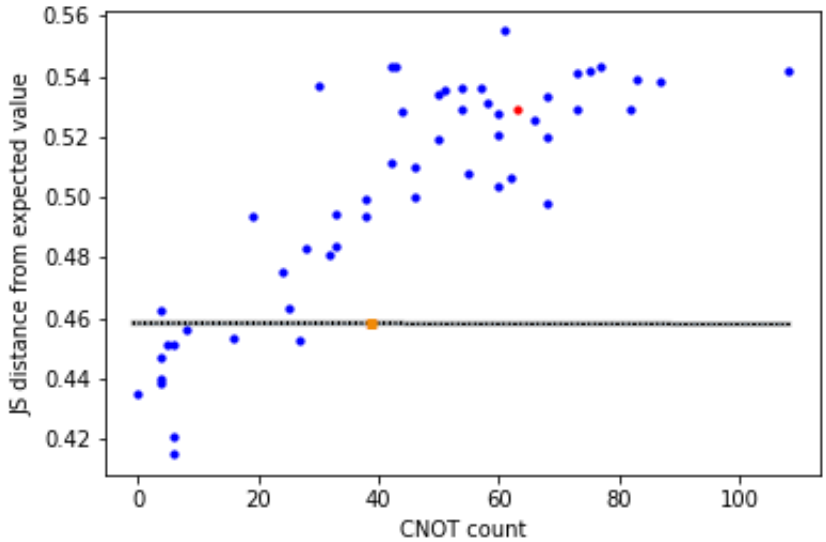


Figure 3.19: JS distance over CNOT count of approximate circuits for 4 qubit Toffoli on the Toronto physical machine showing mappings generated by Qiskit with optimization level 3.

CHAPTER

4

COMBINING HARD AND SOFT CONSTRAINTS IN QUANTUM CONSTRAINT-SATISFACTION SYSTEMS

4.1 Introduction

Much like GPUs, which have become omnipresent in high-performance computing (HPC) systems, quantum processing units (QPUs) are intended to accelerate computational kernels. The difference is that QPUs offer the potential of solving computationally hard problems in shorter time than would be possible via *any* form of classical computing—either by a constant though polynomial factor (termed "quantum advantage") or in a few cases even exponentially, making classically intractable problems tractable (termed "quantum supremacy"). In today's age of noisy, intermediate-scale quantum (NISQ) computation [Preskill, 2018], practical experiments are limited by the number of available qubits and their high susceptibility to noise. Consequently, quantum supremacy in particular has been demonstrated to date on actual QPUs only for problems or input sizes that lack practical applicability [Arute et al., 2019, Pednault et al., 2019, Aaronson, 2019]. Nevertheless, the hope that future, fault-tolerant quantum computers will usher in a new era of HPC makes quantum computing an area with significant

research potential and relevance to the HPC community.

Quantum programming requires a way of thinking that is very unlike that of classical programming and as such can have a high barrier of entry even for those already comfortable with coding in a variety of classical computer languages. Furthermore, there is substantial architectural variety among different quantum computers—analogous to CPUs vs. GPUs vs. TPUs [Jouppi et al., 2018] vs. IPUs [Louw and McIntosh-Smith, 2021] vs. RDUs [Prabhakar et al., 2022] and the like in the classical world—which frustrates the creation of a portable programming model.

Currently, the two dominant architectural models for quantum computers are the *circuit model* and the *annealing model*. Most hardware vendors, including IBM, IonQ, Rigetti, Honeywell, ColdQuanta, PsiQuantum, Quantum Brilliance, and many more, are basing their products on the circuit model [Clarke and Wilhelm, 2008, Cirac and Zoller, 1995, ibm0, , rig, 2022, Wright et al., 2019]. At its core, a circuit-model program is an enormous $(2^n \times 2^n)$ unitary matrix, expressed as the product of tensor products of small (usually 2×2 and 4×4) unitary matrices.

D-Wave [Boothby et al., 2021] is the lone vendor championing the annealing model, although Fujitsu's Digital Annealer [Aramon et al., 2019] represents a classical analogue (same computational model but a classical rather than a quantum implementation). Although both the circuit model and the annealing model are ultimately governed by the Schrödinger equation, an annealing-model program is essentially a quadratic pseudo-Boolean function. The hardware searches (heuristically) for the inputs that minimize this function [Boixo et al., 2012, Bacon et al., 2013].

Being tied specifically to a particular type of optimization problem, the annealing model is more restrictive than the general-purpose circuit model. However, the annealing model offers an important engineering advantage: scalability. D-Wave has manufactured annealing devices with about two orders of magnitude more qubits than what is available today for the circuit model. At the time of this writing, D-Wave's largest machine provides nearly 5,760 qubits, while IBM's largest machine provides only 127.

To date, there have been virtually no attempts to develop a high-level programming model that bridges these two quantum computational models. Because of the popularity of the circuit model, most programming systems target that. Some recent examples of circuit-model programming languages are Twist [Yuan et al., 2022], Silq [Bichsel et al., 2020], Q# [Svore et al., 2018], ProjectQ [Steiger et al., 2018], 𝒬WIRE [Paykin et al., 2017], Scaffold [JavadiAbhari et al., 2015], and Quipper [Green et al., 2013]. D-Wave's Ocean API [D-Wave Systems Inc.,] facilitates the expression of annealing-model programs. All of these work at a fairly low level of abstraction. The circuit-model systems provide mechanisms for juxtaposing small unitary matrices in a

large matrix product, and the annealing-model system provides mechanisms for specifying coefficients for a quadratic pseudo-Boolean function.

A rare example of cross-paradigm quantum programming is NchooseK [Khetawat et al., 2019, Wilson et al., 2021]. NchooseK is a domain-specific language focusing on the domain of constraint satisfaction problems. It seeks to work at a sufficiently high level of abstraction as to both facilitate programming, even for quantum novices, and enable execution on both circuit-model and annealing-model devices. The fundamentals of a simplistic NchooseK abstraction was first used for a Grover search by Khetawat et al. [Khetawat et al., 2019] and developed further for simple constraint-satisfaction problems in a workshop paper by Wilson et al. [Wilson et al., 2021]. Section 4.2 elaborates further, but a small example of an NchooseK program is $nck(\{a,b\},\{0,1\}) \wedge nck(\{b,c\},\{1\})$, which is interpreted as "Neither or exactly one of a and b must be TRUE, and, simultaneously, exactly one of b and c must be TRUE."

This paper presents a more generalized variant of NchooseK for expressing complex constraint-satisfaction problems. Specifically, the paper makes the following contributions:

- It introduces soft constraints—constraints, which, if broken, will incur a penalty but will not invalidate the problem. Soft constraints are crucial for expressing minimization or maximization problems in NchooseK.
- It evaluates a larger set of NchooseK problems, including both hard and soft constraints, than had previous been studied.
- It compares both the complexity of NchooseK and the quality of the *quadratic unconstrained binary optimization* (QUBO) expressions used as an intermediate representation of NchooseK, by comparing them to manually created QUBOs for the same problems.
- It evaluates quantum computations of much larger scale in today's terms than previous work—of up to 65 qubits on the IBM gate-based machines, utilizing every qubit on the ibmq_brooklyn [IBM Quantum Services,], and 1163 qubits on the D-Wave quantum annealers, even within a range where correct answers were potentially no longer found.

4.2 Background

NchooseK is a programming paradigm based on expressing constraint-satisfaction problems over a set of boolean variables. Each constraint in a problem specification takes the form, "Given a variable collection of size N, a specified number of them, K, must be TRUE." Before elaborating we state some relevant definitions:

Definition 1 (Variable collection)**.** A *variable collection* comprises a number of Boolean variables in which variables can be repeated, but order does not matter. Its cardinality is the number of elements (which can exceed the number of unique variables due to repetitions).

Definition 2 (Selection set)**.** A *selection set* comprises a set of disjoint whole numbers, none of which can be greater than the cardinality of a corresponding variable collection.

Definition 3 (Hard constraint)**.** An NchooseK *hard constraint*, written as $nck(N,K)$, consists of a variable collection N and a selection set K. It is satisfied if the cardinality of the variable collection whose variables are TRUE equals one of the numbers in the selection set:

$$nck(N,K) \equiv \left(\sum_{n \in N} n \right) \in K,$$

where $n \in \{0,1\}$ and we associate FALSE with 0 and TRUE with 1.

Definition 4 (NchooseK program)**.** An *NchooseK program* is a conjunction of NchooseK hard constraints written as $nck(N_1,K_1) \wedge nck(N_2,K_2) \wedge \cdots \wedge nck(N_n,K_n)$. The result of executing a program is either an assignment of Boolean values to all variables over the variable collections such that all hard constraints are honored or an indication that no such assignment exists.

To create useful NchooseK constraints, a programmer must focus on the relationships among variables. For example, consider a collection containing the variables a and b. The problem formulation in which a and b must both be TRUE is given by the constraint $nck(\{a,b\},\{2\})$. This indicates that exactly two of a and b must be TRUE, and therefore none can be FALSE. If they need to have the same value but it does not matter which, this would be expressed as $nck(\{a,b\},\{0,2\})$. By including two numbers in the selection set, K, this constraint will be satisfied if two variables are TRUE or if zero variables are TRUE but not if exactly one is TRUE. If, on the other hand, the two variables need to have different values, the constraint would be $nck(\{a,b\},\{1\})$, indicating that exactly one must be TRUE, and, therefore, the other must be FALSE. If at least one of a and b need to be TRUE, the constraint would be $nck(\{a,b\},\{1,2\})$. Omitting 0 from the selection set ensures that they cannot both be FALSE.

As a more complicated example, consider satisfiability problems, discussed more in depth in Section 4.6. A satisfiability problem accepts an expression in conjunctive normal form (conjunctions of unions of possibly negated variables) and reports whether there exists a variable assignment that makes the expression TRUE. "$(v_1 \vee v_2 \vee \neg v_3) \wedge (\neg v_2 \vee \neg v_3 \vee v_4) \stackrel{?}{=} \text{TRUE}$" is an example of a 3-SAT problem, which is a satisfiability problem in which each clause contains at most three variables. For a single 3-SAT clause $(x \vee y \vee z)$ to be TRUE, at least one of the three

variables must be TRUE. This is expressed in NchooseK with the constraint

$$nck(\{x, y, z\}, \{1,2,3\}).$$

This constraint is illustrated graphically in Figure 4.1.

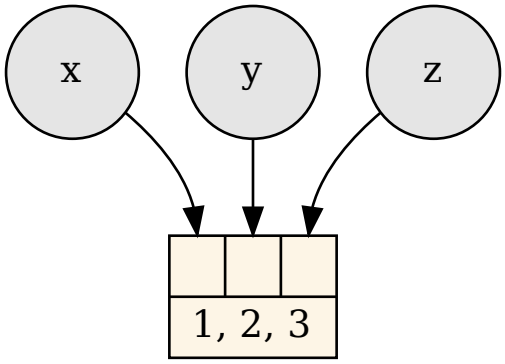


Figure 4.1: A visual representation of a 3-SAT clause with the variables x, y, and z. The nodes represent the Boolean variables, and the box indicates the constraint.

4.3 Related Work

A number of quantum circuit languages are being developed, either as standalone languages or as embedded domain-specific languages. These include Q# [Svore et al., 2018], Twist [Yuan et al., 2022], Silq [Bichsel et al., 2020], ProjectQ [Steiger et al., 2018], 𝒬WIRE [Paykin et al., 2017], Scaffold [JavadiAbhari et al., 2015], and Quipper [Green et al., 2013]. D-Wave's Ocean API [D-Wave Systems Inc.,] likewise functions as a language for their annealing devices and simulators. While not a language per se, Xanadu's PennyLane is a quantum machine-learning software package designed to work across a number of circuit-model systems [Bergholm et al., 2018]. In contrast to those efforts, which target a single computational model apiece, NchooseK programs run unmodified on both annealing-model and circuit-model machines.

XACC [McCaskey et al., 2020] is a software infrastructure that can interface to multiple hardware platforms, including both circuit-model and annealing-model systems. It enables classical programs to embed blocks of quantum code, e.g., written in Quil [Smith et al., 2017], and designate a quantum computer on which to run it. The primary difference with NchooseK is that NchooseK raises the level of abstraction above that of the underlying form of quantum computation, enabling true portability across computational models. XACC, in contrast,

enables a program to integrate circuit-model-specific code that runs only on circuit-model quantum computers and annealing-model-specific code that runs only on quantum annealers. Despite defining its own intermediate representation, XACC is not designed to run any given piece of code on both circuit-model quantum computers and quantum annealers. Another difference between the two systems is that one can program in NchooseK without any knowledge of quantum computing while XACC programmers must be familiar with at least one quantum computational model.

The closest related work to ours is Wilson et al. [Wilson et al., 2021], which introduces NchooseK for hard constraints. However, their work lacks soft constraints, which are essential for generalizing NchooseK's applicability to maximization and minimization problems. Our work not only fills this gap but also presents a problem complexity analysis, considers symmetrical constraints in doing so, and more thoroughly evaluates success characteristics through an empirical study involving both quantum circuit and annealing devices.

4.4 Soft Constraints

In this work we propose a generalized NchooseK model that additionally supports *soft* constraints: constraints whose satisfaction is desired but not required. To motivate the need for soft constraints we consider an example of a problem that cannot be expressed in the existing NchooseK paradigm. We attempt to solve this problem first using only hard constraints and then, after showing how that fails, including soft constraints to make the problem expressible.

4.4.1 Problem requirements and initial formulation

Minimum Vertex Cover is a well-known graph problem: Given an undirected graph $G = (V, E)$, a *vertex cover* is a subset of vertices $W \subseteq V$ such that each edge in E is connected to at least one member of W. The Minimum Vertex Cover is the smallest W in cardinality that meets this requirement.

The first step in solving any NchooseK problem is deciding what the variables should represent. Because the solution to a Minimum Vertex Cover problem is formulated in terms of vertices, we associate one variable per vertex such that the variable is TRUE if and only if the corresponding vertex is in W.

4.4.2 Setting up the vertex cover

As a running example, consider the graph in Figure 4.2, which has five vertices and five edges.

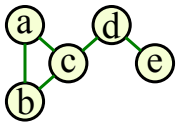


Figure 4.2: A graph of 5 vertices for reference

Consider first a smallest possible subgraph, e.g., the graph $G' = (\{a,b\},\{(a,b)\})$. For G', we can easily determine a minimum vertex cover immediately by expressing the problem with the constraint $nck(\{a,b\},\{1\})$. This ensures that exactly one of the two variables will be TRUE and gives W a cardinality of 1.

An inductive step is non-trivial. If we add to G' vertex c and edges (a,c) and (b,c), the resulting constraints, $nck(\{a,c\},\{1\})$ and $nck(\{b,c\},\{1\})$, cannot both be satisfied. For instance, if $a \in W$ then a is TRUE. In this case, b and c must both be FALSE by the constraints $nck(\{a,b\},\{1\})$ and $nck(\{a,c\},\{1\})$, which ensure that exactly one of the variables in the collections is TRUE, and a must have the same value in all NchooseK constraints within the same program. This leaves the constraint $nck(\{b,c\},\{1\})$ unsatisfiable.

Instead, we need to refine our original constraint to allow both variables to be TRUE if necessary. Using $nck(\{a,b\},\{1,2\})$, as illustrated in Figure 4.3, not only expresses a constraint that finds a vertex cover for our minimal subgraph but can be extended over the entire graph to ensure that a solution can be found.

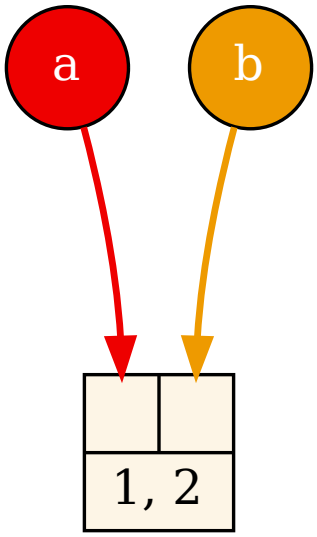


Figure 4.3: A single edge in a vertex cover. Each node corresponds to a vertex in the original graph and a variable in the NchooseK program. The box represents an NchooseK constraint.

The refined NchooseK program for five vertices is

$$nck(\{a,b\},\{1,2\}) \wedge nck(\{a,c\},\{1,2\}) \wedge nck(\{b,c\},\{1,2\}) \wedge nck(\{c,d\},\{1,2\}) \wedge nck(\{d,e\},\{1,2\}),$$

and this is illustrated in Figure 4.4. Unfortunately, this program is incorrect in that it will be

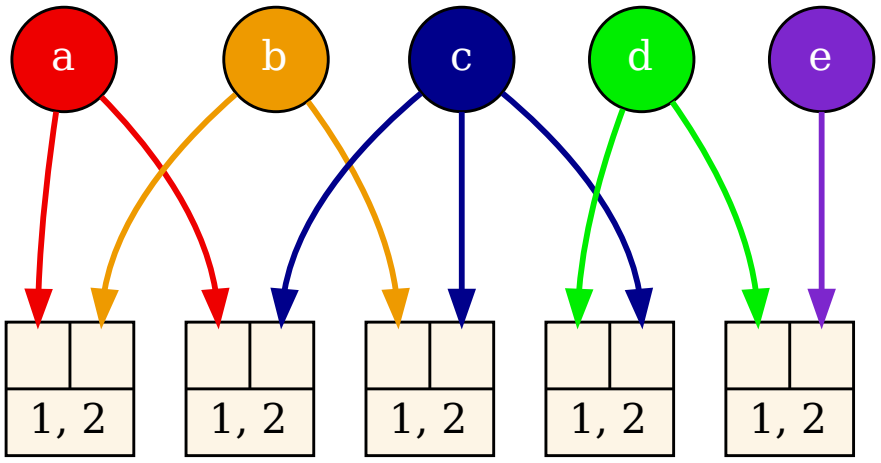


Figure 4.4: A full vertex cover representation in NchooseK. This will be satisfied by every valid vertex cover.

satisfied by *any* vertex cover of the graph in Figure 4.2, not necessarily a minimum vertex cover. The problem is that NchooseK requires all constraints to be met, but this is not generally possible in a minimization or maximization problem.

4.4.3 Minimization via soft constraints

To find specifically a *minimum* vertex cover we propose generalizing NchooseK to support soft constraints in addition to its existing hard constraints:

Definition 5 (Soft constraint)**.** An NchooseK soft constraint, written as $nck(N,K,soft)$, acts as a desired but not required constraint.

Definition 6 (Generalized NchooseK program)**.** A *generalized NchooseK program* is a conjunction of NchooseK hard and soft constraints written as $nck(N_1,K_1) \wedge nck(N_2,K_2) \wedge nck(N_i,K_i) \wedge nck(N_{i+1},K_{i+1},soft) \wedge nck(N_{i+2},K_{i+2},soft) \wedge nck(N_m,K_m,soft)$. The result of executing a program is either an assignment of Boolean values to all variables over the variable collections such that all hard constraint are honored and the number of satisfied soft constraints is maximized; or an indication that no such assignment exists.

In short, the semantics is that an NchooseK program execution will satisfy all hard constraints (or fail if this is not possible) and as many soft constraints as it can.

For a minimization problem, one wants as few variables as possible to be TRUE. To this end, one can associate a soft constraint with each variable: $nck(\{v\},\{0\},soft)$, to indicate a preference but not a demand that v be 0. Consequently, adding the following constraints to our Minimum Vertex Cover program requests that the solution represent a minimum vertex cover:

$$nck(\{a\},\{0\},soft) \wedge nck(\{b\},\{0\},soft) \wedge nck(\{c\},\{0\},soft) \wedge nck(\{d\},\{0\},soft) \wedge nck(\{e\},\{0\},soft)$$

The resulting Minimum Vertex Cover program is illustrated in Figure 4.5.

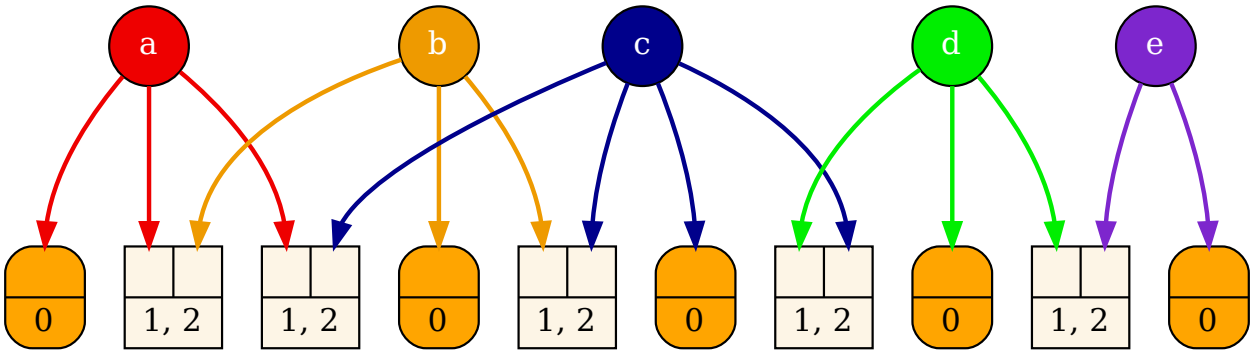


Figure 4.5: A visual representation of a minimum vertex cover represented in NchooseK. The filled boxes with rounded corners are soft constraints and act to minimize the number of vertices in the cover.

Conversely, if one wanted to maximize the variables in a particular problem, one could incorporate a constraint with a selection set of one, i.e., $nck(\{v\},\{1\},soft)$. Constraints like this are among the most common soft constraints used in solving minimization or maximization problems, but they can take other forms as well, potentially opening up problems to more efficient solutions. For example, with the Max Cut problem, one solution is to add an extra variable per edge which is set up to be TRUE if and only if the edge has been cut, then add a soft maximization constraint to each of these new variables. This works, but adds many unnecessary variables and greatly increases the number and complexity of constraints. Another option is to instead have a soft constraint of $nck(\{u,v\},\{1\},soft)$ to every edge. This expresses a preference that every edge be cut, and NchooseK attempts to maximize the number of soft constraints which have been met. This solves the Max Cut problem more efficiently.

4.5 Implementation

One of NchooseK's design goals is to run problems on both circuit-model devices and annealing-model devices. The implementation of NchooseK uses a *quadratic unconstrained binary optimization* (QUBO) format as an intermediate representation. A QUBO seeks to minimize a quadratic equation in which every term comprises either one or two binary variables and a real, constant coefficient. These equations are of the form

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} a_i x_i + \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} b_{i,j} x_i x_j \,, \tag{4.1}$$

and the objective is to find a set of values for variables $\boldsymbol{x} = x_1, ..., x_n$ that minimize $f(\boldsymbol{x})$.

The challenge in creating QUBOs is determining a_i and $b_{i,j}$ coefficients such that the values of $\boldsymbol{x}$ that minimize $f(\boldsymbol{x})$ correspond to the constraints of the target problem. One feature of QUBOs that facilitates the identification of appropriate coefficients is that QUBOs are compositional with respect to addition. If a problem can be broken into small parts before being translated into simple QUBOs, those QUBOs can be combined via addition to form an overall problem QUBO. NchooseK exploits this property by translating each *nck* constraint individually to a QUBO, using QUBO variables to represent the NchooseK variables, before summing all of them into a final QUBO. The NchooseK implementation finds the coefficients of each per-constraint QUBO by expressing the coefficients in terms of a satisfiability modulo theories (SMT) problem, which it then solves using the Z3 SMT solver [de Moura and Bjørner, 2008].

Once an NchooseK program has been compiled to a QUBO, it can be run essentially natively on quantum annealers. NchooseK targets D-Wave quantum annealers by passing the QUBO directly to D-Wave's Ocean API [D-Wave Systems, Inc., a]. For circuit-model devices, NchooseK expresses the QUBO as a problem Hamiltonian suitable for use with the QAOA [Farhi et al., 2014] algorithm—a software analogue of the quantum-annealing process. To run on IBM Q circuit-model quantum computers, NchooseK currently invokes the QAOA function provided by IBM's Qiskit library [Aleksandrowicz et al., 2019]. In either case, each QUBO variable and therefore each NchooseK variable is represented by a qubit, with the state of that qubit corresponding to the value of the variable in the solution. Both of these types of machines may also use additional qubits; this is discussed in more detail in Section 4.8.

As an example, consider the (a, b) edge from the minimum vertex cover, constrained by $nck(\{a, b\}, \{1, 2\})$. We translate this constraint to

$$f(a, b) = a\,b - a - b \,, \quad a, b \in \{0, 1\} \tag{4.2}$$

which is minimized when at least one of a or b has a value of 1. If both edges (a,b) and (b,c) are constrained with $nck(\{a,b\},\{1,2\}) \wedge nck(\{b,c\},\{1,2\})$, this expression will be transformed into $f(a,b)+f(b,c)=(ab-a-b)+(bc-b-c)$, which in turn is minimized over a, b, and c.

Soft constraints introduce additional complexity to the implementation. There is no inherent distinction between hard and soft constraints in QUBOs. To incorporate soft constraints in NchooseK we consider another property of QUBOs: a QUBO function can be scaled by any positive real-valued factor without altering the values that minimize it. However, when multiple QUBOs are combined, larger-in-magnitude coefficients bias the solution towards minimizing those coefficients' associated variables over the variables associated with smaller-in-magnitude coefficients.

We exploit this property in order to strengthen hard constraints over soft constraints. When creating the QUBO for a hard constraint, we multiply its coefficients by a factor of one higher than the total weight of all soft constraints. Doing so ensures that meeting a single hard constraint reduces the overall value of $f(\boldsymbol{x})$ more than would meeting *all* soft constraints. Nevertheless, the more soft constraints are satisfied, the more $f(\boldsymbol{x})$ is further reduced beyond its value from satisfying hard constraints alone.

```
import nchoosek

env = nchoosek.Environment()
verts = ['a', 'b', 'c', 'd', 'e']
edges = [['a', 'b'], ['a', 'c'], ['b', 'c'],
         ['c', 'd'], ['d', 'e']]
for vert in verts:
    env.register_port(vert)
    env.nck([vert], {0}, soft=True)
for edge in edges:
    env.nck([edge[0], edge[1]], {1, 2})
print(env.solve())
```

Figure 4.6: An NchooseK program to solve the minimum vertex cover for the graph shown in Figure 4.2.

NchooseK is implemented as an embedded domain specific language written in Python. Figure 4.6 shows the final vertex cover from Figure 4.5 as a runnable program. Other problems have a similar code structure: the environment is set up, each variable needs to be registered, then each constraint is added with the same syntax as described in this paper. When executed,

Table 4.1: Sample problems, each listed with its complexity class (NP-complete or NP-hard), number of non-symmetric (different types of) constraints, total number of constraints, and number of terms if expressed directly as a QUBO. For Exact Cover and Minimum Set Cover, n refers to the number of the original elements and N refers to the number of subsets.

Problem	Class	# non-symm. constraints	NchooseK constraints	QUBO terms
1. Exact Cover	NP-C	n	n	nN^2
2. Min. Cover	NP-H	n	nN	nN^2
3. Min. Vert. Cover	NP-H	2	$\|V\|+\|E\|$	$\|V\|+\|E\|$
4. Map Color	NP-C	2	$\|V\|+\|E\|n$	$\|V\|n^2+\|E\|n$
5. Clique Cover	NP-C	2	$n\|V\|^2-\|E\|$	$n\|V\|^2-\|E\|$
6. k-SAT	NP-C	2	$n+m$	nm^2+n^2m
7. Max. Cut	NP-H	1	$\|E\|$	$\|E\|+\|V\|$

this program produces the following QUBO:

$$f(a,b,c,d,e) = -11a - 11b - 17c - 11d - 5e + 6ab + 6ac + 6bc + 6cd + 6de$$

This QUBO is isomorphic in the term structure to what one might create by hand, up to the choice of coefficients, which could be chosen differently, e.g., by multiplying by a common positive, real-valued factor.

4.6 Complexity Comparison

NchooseK is intended to be more programmer-friendly than lower-level computational models. We therefore compare the complexity of constructing a problem using NchooseK constraints versus directly constructing a QUBO, which is how one would normally program a quantum annealer or set up a QAOA problem for a circuit-model quantum computer. The set of problems considered is summarized in Table 4.1. Besides distinguishing the complexity class of problems in column 2 (NP-hard and NP-complete), we assess at the number of non-symmetric constraints (column 3) to demonstrate the simplicity of setting up a problem using NchooseK as opposed the less intuitive and error-prone task of formulating a QUBO with changing coefficients dependent on problem size. We observe that problems either fall into the group of (a) constant (1 or 2) or (b) linear non-symmetric constraints relative to their input, which illustrates the ease of programming with the NchooseK abstraction.

Definition 7 (Symmetric Constraints)**.** Two NchooseK constraints are considered symmetric

with one another if they have the same selection set and their variable collections have the same cardinality.

For example, the constraints $nck(\{a,b,c\},\{0,2\})$ and $nck(\{b,c,d\},\{0,2\})$ are symmetric, but $nck(\{a,b,c\},\{0,2\})$ and $nck(\{b,c,d\},\{1,2\})$ are non-symmetric, as are $nck(\{a,b,c\},\{0,2\})$ and $nck(\{b,c\},\{1,2\})$.

When simpler to express a problem, we consider two-local Ising Hamiltonians, in which the variables have values of -1 or 1, as opposed to QUBOs, in which the variables have values of 0 or 1. A simple linear transformation maps between the two problem forms.

Columns 4 and 5 indicate the worst-case complexity of problem formulations as NchooseK constraints vs. as QUBOs, respectively. In most cases, the number of constraints generated by NchooseK is lower than the number of equivalent QUBO terms, often reduced by at least one polynomial order with few a few exceptions (minimum cover, clique cover), again a reflection of NchooseK's conciseness as an abstraction.

4.6.1 Number of terms and number of constraints

Exact set cover The exact cover problem, which is NP-complete, is covered in depth in a related workshop paper [Wilson et al., 2021] and will be described only briefly here. Given a set E and a set S of subsets of E, find a subset of S such that every element of E is included exactly once. This can be solved with NchooseK by adding a constraint for each element of E with a variable collection containing a variable corresponding to each subset which contains that element, and a selection set of {1}.

For an exact cover problem with n elements and N subsets, NchooseK requires n constraints, all of which may be non-symmetric and could have a variable collection cardinality of up to N. To formulate the QUBO directly one can adapt the Ising Hamiltonian

$$H_A = A\sum_{\alpha=1}^{n}\left(1-\sum_{i:\alpha\in V_i} x_i\right)^2$$

along the lines of Lucas [Lucas, 2014], where α refers to an element and V_i refers to subset i. The factor A may be omitted ($A=1$) in this context. With this equation, removing constant terms and x_i^2 terms (because $x_i=-1$ or 1, which becomes the constant 1 when squared), we have at least n terms, but realistically would encounter more constraints as a problem where each element is only in one subset would be trivial.

If an element is included in m subsets, however, that element alone would introduce $m(m+1)/2$ terms. This direct formulation has a worst-case complexity of $nN(N+1)/2$ or

$O(nN^2)$ compared to only $O(n)$ for NchooseK. Both formulations have the same best case.

Minimum set cover The minimum set cover is NP-hard and is the same as the exact cover problem with two key differences: each element of E can be in the solution multiple times, and the goal is to find the smallest subset of S which contains every element of E. This needs the same number of constraints using the same variable collections as the exact set cover, with the selection set now containing every positive integer up to the cardinality of the variable collection. It also requires one soft constraint per subset in order to minimize the number of subsets in the cover.

Both NchooseK and the QUBO formulation for this problem are set up initially as in the exact cover, but require n additional terms to express the minimization; the worst-case complexity is therefore unchanged. It should be noted that in this case these additional terms in the QUBO can be combined, but two different coefficients for these terms need to be chosen and balanced against each other.

Minimum vertex cover For the minimum vertex cover, an NP-hard problem described in Section 4.4, the NchooseK solution requires $|E|$ hard constraints and $|V|$ soft constraints. The corresponding Hamiltonian is formulated as the QUBO

$$H = A \sum_{uv \in E} (1-x_u)(1-x_v) + B \sum_{v} x_v$$

where u and v denote vertices. This results in $3|E|+|V|$ terms, the same complexity as NchooseK. The number of mutually non-symmetric constraints for NchooseK is only two; every constraint corresponds either to an edge of the form $nck(\{u,v\},\{1,2\})$ or to a vertex of the form $nck(\{v\},\{0\},soft)$.

Map coloring The map coloring problem with n colors is another NP-complete problem covered in depth by Wilson et al. [Wilson et al., 2021]. The solution uses one-hot encoding, meaning it assigns n variables per vertex, with each variable indicating if the vertex has the associated color. If vertex v has color options 1, 2, and 3, it has variables v_1, v_2, and v_3. If v_1 is TRUE, the other two will be FALSE, and vertex v will have color 1. We need one constraint per vertex to ensure that the vertex has only one color. The variable collection contains n variables, one for each color, and the selection set is $\{1\}$. This problem also requires n constraints per edge. For these constraints, the variable collection contains two variables corresponding to the same color on each of the vertices the edge is connecting. The selection set is $\{0,1\}$, ensuring that two adjacent vertices do not share a color: $nck(\{u_i,v_i\},\{0,1\})$. Every constraint in the map

coloring problem will be symmetric with one of these two types.

Our NchooseK solution therefore requires $|V|+n|E|$ constraints. A QUBO using the same one-hot encoding scheme is

$$\sum_v \left(1-\sum_{i=1}^{n} x_{v,i}\right)^2 + \sum_{(uv)\in E}\sum_{i=1}^{n} x_{u,i}\,x_{v,j}$$

This uses $|V|n/2(n+1)+|E|n$ terms, leading to $O(|V|n^2+|E|n)$ compared to NchooseK's $O(|V|+|E|n)$. This same trend is seen any time one-hot encoding is used; if n designations are used between V vertices, the QUBO results in $O(Vn^2)$ terms while NchooseK uses only $O(V)$ constraints.

Clique cover The clique cover problem is NP-complete. It requires the coloring of a graph with n colors such that the nodes of each color form a clique within the color. As in the map coloring problem, the solution to this problem requires one-hot encoding with one constraint per vertex. It also needs n constraints per edge *absent* from the graph to ensure that two vertices that are not adjacent do not share a color, similar to the constraints in the map coloring problem. It also needs only two types of non-symmetric constraints.

The clique cover solutions are nearly identical in terms of NchooseK constraints and QUBO terms. Both depend on the number of possible edges not included in E. This is enumerated as $|V|(|V|-1)/2-|E|$. In both cases, the solution requires $O(n|V|^2-|E|)$ terms or constraints.

k-satisfiability The NP-complete k-satisfiability problem establishes m constraints over n boolean variables, each constraint of cardinality k. One or more variables per constraint must have the value of either TRUE or FALSE specified by the constraint. This is similar to how NchooseK constraints are built, with one major exception: NchooseK requires either twice as many variables or much more complicated constraints. The satisfiability constraints can force variables to be either TRUE or FALSE in their constraints without treating them any differently, but NchooseK does not have that capability.

One solution is to create one ancilla variable per original variable, where the ancilla has the opposite value, for example x and $\neg x$. These need a constraint to ensure that they have opposite values, with a selection set of $\{1\}$. Furthermore, one constraint is required per satisfiability constraint with the same variables in the variable collection. The selection set contains every positive integer up to and including k, as seen in Figure 4.1 for 3-SAT. Using this solution, two non-symmetric types of constraints are used.

The other solution is to create more complicated constraints. Variables can be treated dif-

ferently from one another by inserting additional copies of them in the variable collection. For the satisfiability constraint $\{x, y, \neg z\}$, the NchooseK specification $nck(\{x, y, z, z, z\}, \{0, 1, 2, 4, 5\})$ establishes the same constraint, as all instances of z must have the same value. This approach requires fewer NchooseK variables and fewer constraints, but the more complicated constraints run the risk of requiring more ancillary qubits. Copying variables in this manner also changes the number of non-symmetric constraints, giving us a worst case of k. Copying variables further impedes simplicity of expression, which motivated the creation of NchooseK in first place.

When considering its complexity, the dual variable setup of NchooseK for a satisfaction problem with n variables and m constraints requires $n+m$ constraints, while the same problem with larger variable collections requires only m constraints. QUBO formulation of this problem is more complicated. One common solution translates the 3-SAT problem into a Maximum Independent Set problem [Choi, 2011, Gabor et al., 2019, Lucas, 2014]. This requires km variables, one variable for each variable within each constraint and one term per variable. $k(k-1)m/2$ terms are required between the variables within constraints. Additional terms result from each instance of TRUE/FALSE versions of the variables—if there are i constraints with x and j with $\neg x$, ij terms would be needed to ensure that a variable never has more than one value. In the worst case, this amounts to $m^2k/4$, giving the QUBO a worst-case complexity of $O(km^2+k^2m)$, compared to the NchooseK worst case of $O(n+m)$.

Maximum cut The NP-hard max cut problem is one of the simplest to express in NchooseK: only one soft constraint is needed per edge. The variable collection contains the vertices of the edge, and the selection set is {1}. These soft constraints ensure that as many vertices as possible have the opposite value to their adjacent vertices. All constraints are symmetric with one another. The max cut problem produces an equal number of NchooseK constraints and Ising terms: $O(|E|)$. However, conversion from Ising to QUBO increases the complexity to $O(|E|+|V|)$ for this particular problem.

4.6.2 Generated versus manually produced QUBOs

As NchooseK translates to QUBOs before solving on both gate-based and annealing devices, an important question then is how these translated QUBOs compare to handcrafted QUBOs for the same problem.

QUBO creation is itself computationally difficult. NchooseK uses the Z3 SMT solver [de Moura and Bjørner, 2008] to map an individual constraint to a QUBO. For every problem discussed in this paper with the exception of the satisfaction problem and minimum set cover, the QUBO used in NchooseK is the same as the handcrafted QUBO for that problem. This holds regardless

of problem size for three reasons:

- NchooseK converts each constraint individually. In most of the problems discussed here, extending the problem means adding additional symmetric constraints (e.g., $nck(\{a,b\},\{0,1\})$ and $nck(\{c,d\},\{0,1\})$). These additional constraints will be converted to QUBOs with the same performance as the previous ones.
- QUBOs are compositional. Two constraints which have been converted into QUBOs are combined with simple addition, meaning that the number of constraints used has no effect on the efficacy of the conversion.
- Constraints with a selection set of {1} are trivial to convert to a QUBO, even for large variable collections. No efficacy of conversion is lost for those problems in which extending the problem likewise extends the size of the variable collection, such as adding additional colors in the map coloring problem or subsets in the exact cover problem.

Discussion Many problems require the introduction of ancillary variables to enable their expression as a QUBO. For example, the NchooseK constraint $nck(\{a,b,c\},\{1,3\})$ cannot be expressed as a three-variable QUBO; it requires a fourth, ancillary variable for an additional degree of freedom in computing the QUBO coefficients. In the minimum set cover problem, constraints with a large variable collection and a large selection set will occasionally have ancillary variables added, whereas there are none in the handmade QUBO for the same problem. Even in this case, the number of additional terms is upper-bounded by $O(nN^2)$. Satisfiability problems exhibit a similar difference in the number of ancillary variables between NchooseK and handmade QUBOs.

4.6.3 Ease of construction

Setting up a problem in NchooseK is simpler and more intuitive than setting up the same problem directly as a QUBO even though the number of NchooseK constraints is often similar to the number of QUBO terms. This is due to the fact that constraints are often symmetric across variable sets, and their corresponding selection sets correspond to the problem specification. That is, a constant number of constraint forms tend to be replicated over variable permutations. In contrast, QUBO coefficients change as problem sizes change, and for some constraints ancillary variables may be required. It is not apparent from a problem formulation how many ancillary variables, if any, will be required.

Wilson et al. [Wilson et al., 2021] examine the difference in creating an NchooseK and a QUBO for the equation $A \oplus B = C$. We reiterate their conclusions here: To write an XOR

equation $c = a \oplus b$ in NchooseK, the constraint $nck(\{a,b,c\},\{0,2\})$ can easily be obtained by inspection of the XOR truth table. To write the same equation as a QUBO, a number of algebraic transformations are needed. In addition, this equation requires an ancillary variable. The final QUBO is given by

$$f(a,b,c,\kappa) \;=\; a \;+\; b \;+\; c \;+\; 4\kappa \;-\; 2ab \;-\; 2ac \;-\; 4a\kappa \;-\; 2bc \;-\; 4b\kappa \;+\; 4c\kappa \quad , \quad (4.3)$$

where κ is an ancillary variable without which $f(a,b,c)$ cannot be expressed as a QUBO.

Not only are QUBOs difficult to create by hand, but, as is apparent from Eq. 4.3, QUBOs are also not particularly human-readable. This is especially true when ancillary variables are used. Compared to $nck(\{a,b,c\},\{0,2\})$, Eq. 4.3 is complex and obtuse.

4.7 Experimental Setup

We ran a variety of experiments on IBM's 65-qubit circuit-based machine, ibmq_brooklyn [IBM Quantum Services,], and one of D-Wave's annealing machines, Advantage 4.1 [D-Wave Systems, Inc.,]. In the case of the circuit-based machines, running the program relies on preparing a subroutine (a Hamiltonian function known as a "phase separator") for the Quantum Approximate Optimization Algorithm (QAOA) [Farhi et al., 2014]. QAOA sequentially runs multiple circuits—in our case, 4000 times each—which produce a single result. In contrast, the annealing machines run a single circuit multiple times—in our case, 100. Each run produces a result. For the experiments described in this section we consider only the best (lowest-energy) result.

All problems in Section 4.6 are either NP-hard or NP-complete. They fall under three categories. (1) Problems exclusively with soft constraints (NP-hard): max cut; (2) Problems with a mix of hard and soft constraints (NP-hard): minimum vertex cover and minimum set cover; (3) Problems exclusively with hard constraints (NP-complete): clique cover, map coloring, satisfaction, and exact cover. Of these problems, only those without soft constraints could be solved by the original NchooseK abstraction prior to us adding soft constraints in this paper, and only map coloring and exact cover had been discussed in prior NchooseK work [Wilson et al., 2021] and only for small problems.

In the world of classical computing, metrics tend to focus on execution time. In contrast, the noise of contemporary quantum devices forces researchers to assess *which, if any, of the provided answers are correct in the first place*. To this end, we establish the following terminology for NchooseK:

Definition 8 (Optimal, suboptimal and incorrect)**.** An NchooseK solution over h hard and s soft constraints is *optimal* if all hard and as many soft constraints as possible are satisfied; it is

suboptimal if all hard (but less than maximum soft) constraints are satisfied; and it is *incorrect* if fewer than h hard constrains are satisfied.

The rationale here is that for problems only using hard constraints, an optimal solution requires full constraint satisfaction, but more than one optimal result may exist. For mixed hard/soft problems, suboptimal solutions still meet all hard constraints but not the maximum number of soft ones, which provides a solution that can be considered non-minimal.

We determined if the results with soft constraints were optimal by checking against the Z3 solver, which solves the problems classically. For mixed problems run on ibmq_brooklyn, results were optimal at smaller scale before becoming suboptimal and then incorrect at larger scale. That is, there seems to be a discrete barrier to optimal solutions. Exposing the same problems to Advantage 4.1 resulted in more suboptimal solutions than optimal ones. Because we are more interested in optimal solutions, we report how many optimal solutions were found.

Subsequent experiments focus on how complex NchooseK problems can become before only incorrect answers are returned. Scaling up the problems from Section 4.6, we study how the addition of variables and constraints affects the answers obtained. The clique cover problem and map coloring problems require many more qubits than the others. Up to the limit of these two problems, which varies depending on the physical machine, all of the graph problems (Minimum Vertex Cover, Max Cut, Clique Cover, and Map Coloring) are performed on the same graphs.

We ran two different scaling studies: vertex scaling and edge scaling. For vertex scaling, each iteration adds a clique of three vertices connected to the previous iteration by two edges up to 33 vertices. After 33 vertices the scaling continues in larger increments until the max cut and minimum vertex cover problems use all of the qubits on the IBM machine, and correct (optimal/suboptimal) results are no longer found on the D-Wave system.

For edge scaling, 12 vertices are used—this is where the clique cover problem fails on the D-Wave system. The first one to fail under vertex scaling is the clique cover problem on Advantage 4.1. This problem initially has four cliques and 18 edges. Six or seven edges are added each time up until 48 edges, where adding a single edge between any two disconnected vertices would allow it to be covered by only three cliques. More edges are then added up until 63 edges, at which point adding another edge would allow it to be covered by only two cliques. In this region, the clique cover problem is run with a target of both three and four cliques for comparison.

For the exact cover, minimum set cover, and satisfaction problems, each problem is generated randomly in increasing size with the exact cover and minimum set cover using the same sets and subsets. The k-satisfiability problems are all 3-SAT problems, i.e., every satisfiability constraint contains three terms. The same problems are run on each type of machine.

4.8 Results

4.8.1 D-Wave Advantage 4.1

Figure 4.7 presents measurements the percentage of results (y axis) that are optimal (as opposed to suboptimal or incorrect) over the number of qubits (x axis) on the D-Wave system. With the exception of the exact set problem, the problems with soft constraints generally perform worse than problems exclusively using hard constraints. This is due to the fact that in mixed problems hard constraints receive a higher bias (in terms of constraint factors) than soft constraints. This makes the energy gap relatively small between one solution and another with an additional soft constraint satisfied. If we, instead, reported the percentage of optimal *and* suboptimal results in the y axis, mixed problems would have a higher success rate (omitted due to space). We also observed that the total number of optimal+suboptimal solutions for mixed problems is larger than the number of optimal solutions for hard ones using similar numbers of qubits.

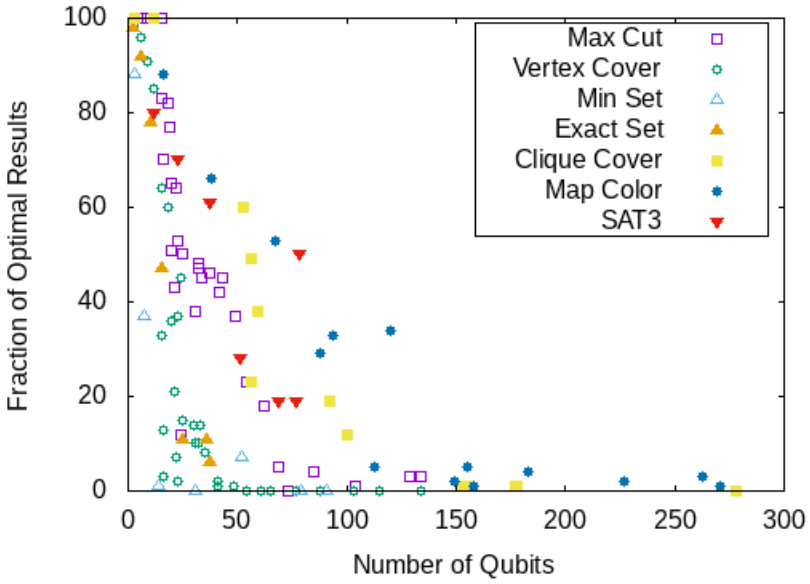


Figure 4.7: Fraction of optimal results on D-Wave systems versus number of qubits.

The number of qubits and the connectivity between them for D-Wave's annealing devices are important considerations. First, the Advantage 4.1 system has 5,640 qubits so any problem that requires more will not be able to be run on that machine. Second, problem variables (e.g., nodes of a graph) are often coupled to many other variables. Given the physical qubit graph topology of a D-Wave device, a variable may need to be mapped to a *chain* of qubits to

establish these couplings. Hence, the more densely connected the problem, the more qubits are required to represent each variable. This ratio tends to become significant for larger problems.

This explains why the number of qubits used on D-Wave systems relates not only to the number of NchooseK variables used, but also to the number of constraints, which affect the number of connections needed on the physical annealing device. For the clique cover, 48 variables and 18 edges requires 188 qubits, but increasing the number of edges *reduces* the number of constraints for this particular problem formulation. For 37 edges, optimal results are found again as only 132 qubits are needed. At the extreme of 63 edges, still using 48 variables, only 52 qubits are used, increasing the success rate to 65%.

In fact, reducing the number of constraints can have as great an effect on the accuracy as reducing the number of variables does. For the clique cover again with 48 variables, increasing the number of constraints from 24 to 36 results in a drop in success rate from 65% to 20%. These solutions use 52 and 55 qubits, respectively, i.e., only a small increase in the number of qubits is imposed. Instead, if we use 27 variables and 78 constraints, 57 qubits are required with a success rate of just 39%. Decreasing the number of variables used from 48 to 27 still results in a significant drop in success rate because the number of constraints increases dramatically, even though the number of qubits used is similar.

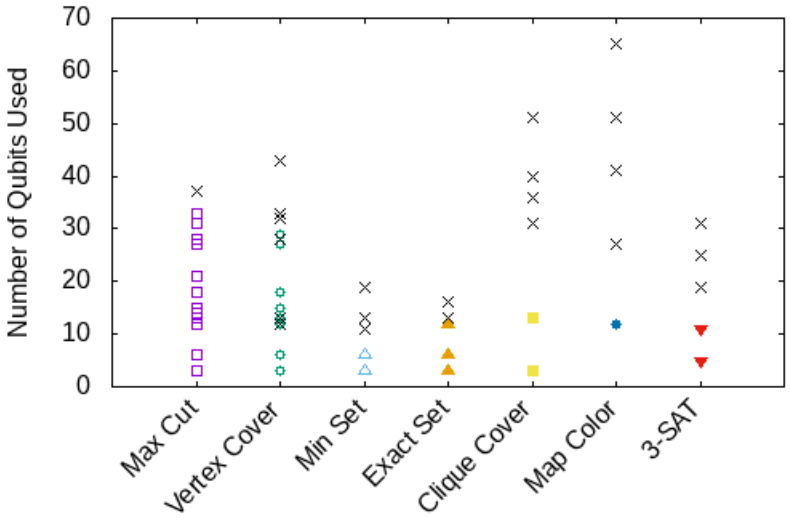


Figure 4.8: Optimal (colored tics) and suboptimal or incorrect (block × tics) results of the QAOA problems for ibmq_brooklyn vs. number of qubits used.

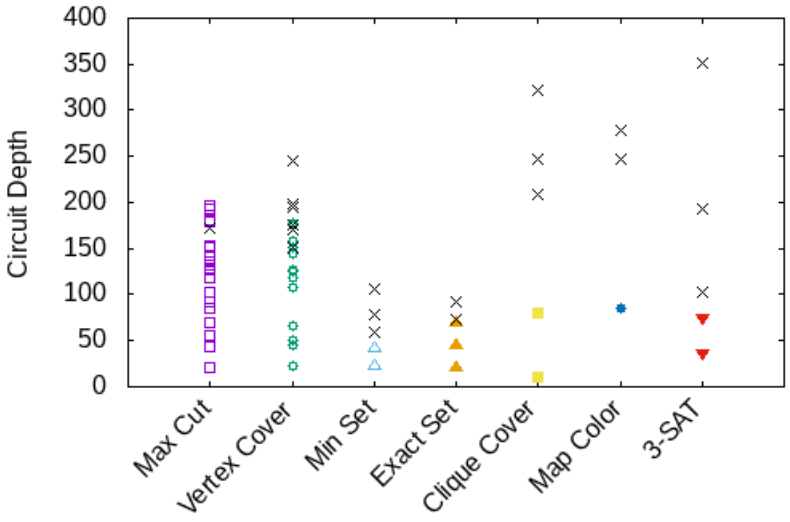


Figure 4.9: Optimal (colored tics) and suboptimal or incorrect (block × tics) results of the QAOA problems for ibmq_brooklyn vs. circuit depth. Six failed clique cover problems were omitted for clarity; they used circuits of depth 432, 516, 537, 676, 697, and 717.

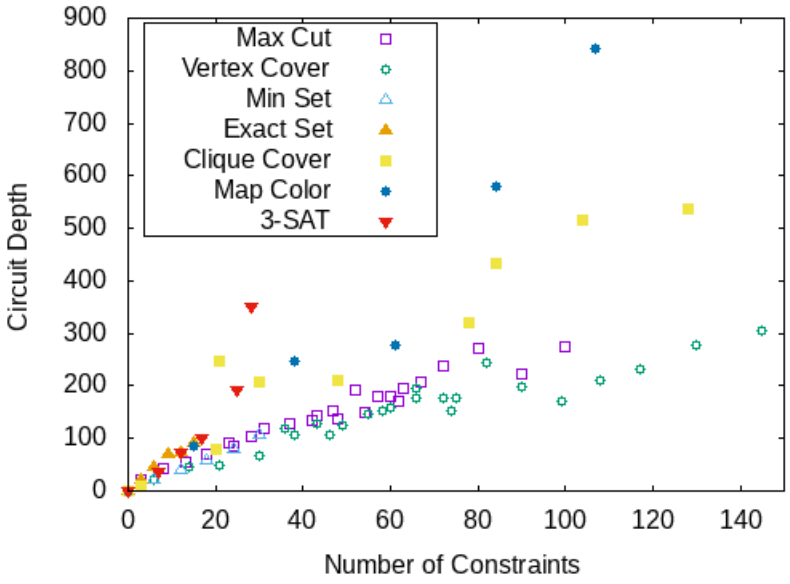


Figure 4.10: The depth of QAOA circuits with respect to the number of constraints in the NchooseK problem.

4.8.2 IBM Q Brooklyn

The problems performed worse on ibmq_brooklyn than on Advantage 4.1; different problems failed to find an optimal result at a lower number of variables and constraints than used for annealing. Despite this, it should be stressed that using QAOA a single result is returned and found to be optimal or not, while using an annealer the problem is considered to be solved correctly if any of the hundred solutions returned is optimal.

As with annealing devices, the number of qubits is an important consideration when utilizing circuit-model devices. The most obvious reason is that the machine has far fewer qubits; no NchooseK problem with more than 65 variables can be mapped onto ibmq_brooklyn. Another factor is that some qubits and some connections between qubits are worse than others in terms of noise. Small problems may select the best performing qubits on a given device, while larger ones must use more error-prone ones as the fraction of utilized qubits increases. Due to limited qubit connectivity in the physical topology, circuit-model machines cannot directly perform two-qubit operations on arbitrary pairs of qubits. Hence, they must frequently swap the state of adjacent qubits in sequence to move pairwise interactions to physical neighbors. The compiler sometimes prioritizes a shorter but lower-quality (higher-noise) path of swaps. This affects solution quality as the number of qubits and circuit depth increase.

Recall from the discussion in Section 4.6.2 that the QUBO formulation of a problem often requires the introduction of ancillary variables. This explains why the number of qubits sometimes exceeds the number of variables in an NchooseK problem.

Figure 4.8 depicts the number of qubits used (y axis) for problems (x axis) from Table 4.1 indicating both optimal (colored tics) and suboptimal (block × tics) results. We observe that there is a correlation between the number of qubits and obtaining optimal results. Figure 4.9 depicts results for the same programs (x axis) over the circuit depth (y axis) measured as the number of gates in the longest path of a single QAOA circuit with the same tic mark colors as before. While each QAOA runs around 30 different circuits (slight variations are due to convergence properties), these circuits differ by the parameters of the gates (qubit rotation angles), not the type or number of gates. Circuit depth is an important considerations when experimenting with circuit model devices. This is true not only because each gate adds a small amount of probabilistic error (noise) to a circuit, but also because a deeper circuit needs to stay active on the machine longer, leading to an increase in chance of qubits decohering before results can be measured.

These two figures show the trends in correctness for the different problems. Note that the edge study and the vertex study are both included for the map problems. This explains the low qubit failures for the vertex cover seen in Figure 4.8: Even using few qubits, a sufficient number of constraints will add enough complexity to the problem to cause a failure. This

relationship between circuit depth, which can be thought of as a simplistic measure of circuit complexity, and the number of constraints is exposed in Figure 4.10, which depicts the number of constraints (x axis) over circuit depth (y axis) for each problem type. The general trend shows increasing depth as more variables and constraints are added during problem scaling, albeit at different rates per problem, i.e., in a problem-specific manner. Exceptions include the minimum vertex cover: At 30 variables and 82 constraints, it uses 32 qubits with a depth of 245. At 33 variables and 90 constraints, only 33 qubits are used with a depth of 199. Hence, depth is not always related to the success rate (optimality) of results. This was also visible in Figure 4.9, where a suboptimal solution for Max Cut at depth 172 is followed by optimal solutions at 179 and thereafter. Nonetheless, these problems scale up to mid to high teens of qubits on the IBM device (25–100% of qubit utilization) and into the hundreds of qubits on the D-Wave device (4–6% of physical qubit utilization).

4.8.3 Timing

Given the limitations of contemporary quantum computers in terms of qubit counts, coherence times, control precision, resilience to noise, and qubit connectivity, raw execution time is generally not the focus of current quantum-computing research. Nevertheless, we include a brief summary of the time taken and the bottlenecks of running a sample of our problems. The client-side operations for experiments described in this section were performed on a 4 GHz Quad-Core i7 processor with 40 GB memory.

For the problems run on the IBM systems, each execution of the QAOA algorithm implicitly submits approximately 25 to 35 jobs, the number of which does not discernibly depend on the size of the problem. Each job comprised 4000 shots, the default for Qiskit's QAOA, and took between 7 and 23 seconds.

We were unable to determine any correlation between problem size and time per job. Figure 4.11 shows a box plot of job run time (y axis) versus the number of variables (x axis) in the original NchooseK environment.

Aside from the time spent running on the quantum computer, a job also requires computation time on the IBM server. It takes a few seconds to create, transpile, and validate a job plus an indeterminate amount of time waiting in the queue for access to the machine. All together, our jobs spent roughly 500 seconds on IBM's servers, not counting communication or queue time. This time can vary greatly, depending on how full the queue is with unrelated jobs.

On the client side, some amount of time is spent generating the QUBO and working with the optimizer. Relative to the amount of time spent in IBM's cloud, the time spent creating the QUBO is not only negligible—taking a second or less—but is also overshadowed by the

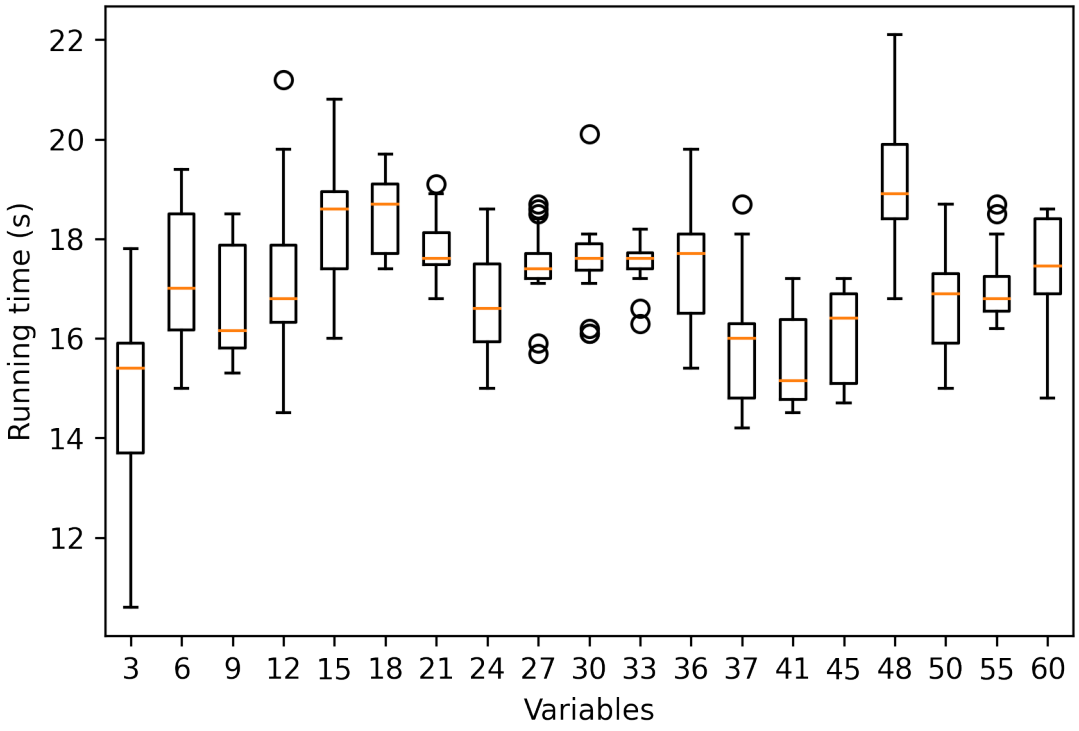


Figure 4.11: The run time of QAOA circuits with respect to the number of variables used.

variance in communication and number of jobs run, not to mention the indeterminate time spent waiting in the job queue. Finally, the classical optimization step in the QAOA process typically takes two to three seconds per job. All together, even a small problem takes about 500 seconds plus queueing time to solve. IBM has recently started offering the option to run QAOA more closely tied to the IBM servers through Qiskit Runtime [Johnson and Ben-Shach, 2022], which should cut down on communication time and possibly time spent on classical optimization.

The problems run on the D-Wave systems were submitted as a single job consisting of 100 samples. According to the D-Wave documentation [D-Wave Systems, Inc., b], each job has a single, relatively long programming step (observed to be on the order of 15ms) in addition to the cost of the 100 samples. The cost of a sample includes the cost of the anneal itself, a parameter that can be defined by the user (our experiments used the default of 20µs); a readout time with a cost that is usually 3–4 times as long as the annealing time; and an added delay between each readout and the subsequent anneal (about 20µs each). The total time for the 100 samples is slightly less than the time than the programming step. Finally, a few more milliseconds are

needed for post-processing. Neglecting the time in the queue, our jobs each spent about 30ms apiece on the Advantage system.

A large cost on the client side is the conversion of constraints to individual QUBOs. This procedure is currently under development and is not yet optimized. Specifically, it redundantly computes QUBOs for symmetric constraints instead of caching previously computed QUBOs. Due to this wasted computation, the total time to compile a complete NchooseK problem to a QUBO is 40–50x the time needed for direct (non-QUBO) solution by the Z3 solver of problems of the size covered in this paper. After constructing the QUBO, preparing it to send to a D-Wave system takes approximately an additional 40ms.

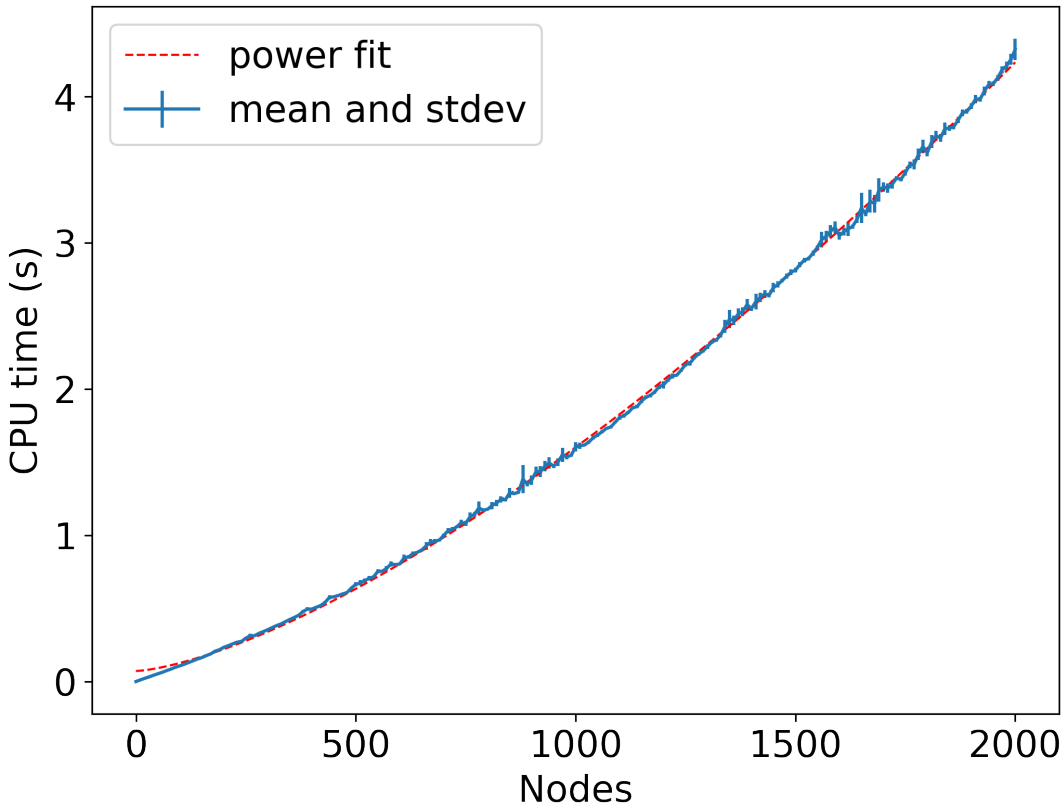


Figure 4.12: The run time of minimum vertex cover on Z3. Each problem was run 30 times on a circulant graph with the indicated number of nodes.

Z3 is a highly optimized classical SMT solver, and it is able to solve each of the problems contained here in less than three seconds. It can also solve problems much larger than can fit on

current quantum hardware, scaling quite well. The minimum vertex cover problems we ran fit very close to a polynomial equation as shown in Figure 4.12. However, when presenting Z3 with problems after they have been translated into a QUBO, many of them perform quite poorly: solving a minimum vertex cover problem with 10 vertices of degree 3 takes less than a second while 20 vertices takes a minute and a half, and 30 vertices takes multiple hours. NchooseK's classical Z3 back end runs faster than either of the two quantum back ends on current quantum hardware. However, we note that the D-Wave Advantage machine completes the optimization step proper in a fraction of a second. This suggests that there exists opportunities to close the performance gap between D-Wave and Z3 through additional software optimizations.

4.9 Future Work

One of the current limitations of NchooseK is its reliance on QAOA for operation on circuit-based machines. We are investigating different methods of converting NchooseK programs into quantum circuits. This may involve abandoning QAOA entirely for an alternative variational quantum algorithm, or it may involve devising NchooseK-specific or problem-specific customizations to QAOA's problem and mixer Hamiltonians. This is the basic concept underlying the Quantum Alternating Operator Ansatz [Hadfield et al., 2019] (a refinement of the Quantum Approximate Optimization Algorithm that is also abbreviated QAOA). The custom mixers used in this version of QAOA seem especially appropriate to NchooseK problems with both hard and soft constraints.

4.10 Conclusions

NchooseK is an effective and relatively simple method of expressing and solving NP-complete problems on both quantum annealers and circuit-based quantum computers. Our contribution is a generalization of NchooseK to include soft constraints, which widens the scope of problems that can be expressed to include NP-hard problems. We show that NP-complete and NP-hard problems can be solved using NchooseK on current, noisy, intermediate-scale quantum (NISQ) devices utilizing up to 65 qubits on IBM's devices and hundreds of qubits on D-Wave's annealing devices. One contribution of NchooseK is given by its intuitive problem formulation with (typically) only a constant or a linear number of non-symmetric constraints, whereas manual QUBO formulations are more complex and require computing different coefficients depending on problem size. Another contribution is that NchooseK enables a transformation even of soft constraints into QUBOs, which make a suitable intermediate representation for

enabling portability across the circuit model and the annealing model. QUBO generation is fully automated, and NchooseK produces QUBOs that are comparable to those painstakingly developed by hand.

Acknowledgments

Research presented in this paper was supported by the Laboratory Directed Research and Development program of Los Alamos National Laboratory under project number 20210397ER. Los Alamos National Laboratory is operated by Triad National Security, LLC for the National Nuclear Security Administration of U.S. Department of Energy (contract no. 89233218CNA000001). This work was also supported in part by LANL subcontract 725530 and by NSF awards DMR-1747426, PHY-1818914, OAC-1917383, MPS-2120757, and CISE-2217020.

CHAPTER

5

CONCLUSION

Quantum computing on real hardware is still in its early stages. Improving the process of using quantum computers is a multidisciplinary undertaking. Even while hardware is being improved, the software has to be developed to improve both the accuracy and accessibility of quantum devices.

In this work, we first demonstrate how software can help overcome the changeability of qubits by measuring their error rates more closely in time to actual circuit execution than is currently standard. We recommend increasing the frequency of error measurement.

Next, we study the feasibility of using approximate circuits in an effort to reduce the amount of noise from qubit decoherance and gate infidelity. We show how, under noise, approximate circuits can outperform their precise counterparts, but that the synthesis and selection of approximate circuits must be further improved before this becomes a viable option.

Finally, we show how software can make quantum computing more accessible with the software package NchooseK. This software both allows certain types of problems to be run on quantum hardware without requiring a steep learning curve while allowing the same problem to be run on both quantum annealers and gate-based quantum computers without additional effort on the part of the programmer.

In summary, we provide evidence toward our hypothesis, as we show that by changing the approach to program quantum computers noise is reduced to more acceptable levels and can

further lower programmatic complexity.

REFERENCES

[rig, 2022] (2022). Welcome to quantum cloud services—QCS documentation.

[Aaronson, 2019] Aaronson, S. (2019). Shtetl-optimized.

[Aharonov et al., 1997] Aharonov, D., Kitaev, A., and Nisan, N. (1997). Quantum circuits with mixed states. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computation (STOC)*, pages 20–30.

[Alam et al., 2020] Alam, M., Ash-Saki, A., and Ghosh, S. (2020). Circuit compilation methodologies for quantum approximate optimization algorithm. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 215–228, Los Alamitos, CA, USA. IEEE Computer Society.

[Aleksandrowicz et al., 2019] Aleksandrowicz, G., Alexander, T., Barkoutsos, P., Bello, L., Ben-Haim, Y., Bucher, D., Cabrera-Hernández, F. J., Carballo-Franquis, J., Chen, A., Chen, C.-F., Chow, J. M., Córcoles-Gonzales, A. D., Cross, A. J., Cross, A., Cruz-Benito, J., Culver, C., González, S. D. L. P., Torre, E. D. L., Ding, D., Dumitrescu, E., Duran, I., Eendebak, P., Everitt, M., Sertage, I. F., Frisch, A., Fuhrer, A., Gambetta, J., Gago, B. G., Gomez-Mosquera, J., Greenberg, D., Hamamura, I., Havlicek, V., Hellmers, J., Herok, Ł., Horii, H., Hu, S., Imamichi, T., Itoko, T., Javadi-Abhari, A., Kanazawa, N., Karazeev, A., Krsulich, K., Liu, P., Luh, Y., Maeng, Y., Marques, M., Martín-Fernández, F. J., McClure, D. T., McKay, D., Meesala, S., Mezzacapo, A., Moll, N., Rodríguez, D. M., Nannicini, G., Nation, P., Ollitrault, P., O'Riordan, L. J., Paik, H., Pérez, J., Phan, A., Pistoia, M., Prutyanov, V., Reuter, M., Rice, J., Davila, A. R., Rudy, R. H. P., Ryu, M., Sathaye, N., Schnabel, C., Schoute, E., Setia, K., Shi, Y., Silva, A., Siraichi, Y., Sivarajah, S., Smolin, J. A., Soeken, M., Takahashi, H., Tavernelli, I., Taylor, C., Taylour, P., Trabing, K., Treinish, M., Turner, W., Vogt-Lee, D., Vuillot, C., Wildstrom, J. A., Wilson, J., Winston, E., Wood, C., Wood, S., Wörner, S., Akhalwaya, I. Y., and Zoufal, C. (2019). Qiskit: An open-source framework for quantum computing.

[Amy et al., 2013] Amy, M., Maslov, D., Mosca, M., and Roetteler, M. (2013). A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(6):818–830.

[Aramon et al., 2019] Aramon, M., Rosenberg, G., Valiante, E., Miyazawa, T., Tamura, H., and Katzgraber, H. G. (2019). Physics-inspired optimization for quadratic unconstrained problems using a digital annealer. *Frontiers in Physics*, 7.

[Arute et al., 2019] Arute, F., Arya, K., Babbush, R., Bacon, D., Bardin, J. C., Barends, R., Biswas, R., Boixo, S., Brandao, F. G. S. L., Buell, D. A., Burkett, B., Chen, Y., Chen, Z., Chiaro, B., Collins, R., Courtney, W., Dunsworth, A., Farhi, E., Foxen, B., Fowler, A., Gidney, C., Giustina, M., Graff, R., Guerin, K., Habegger, S., Harrigan, M. P., Hartmann, M. J., Ho, A., Hoffmann, M., Huang, T., Humble, T. S., Isakov, S. V., Jeffrey, E., Jiang, Z., Kafri, D., Kechedzhi, K., Kelly, J., Klimov, P. V., Knysh, S., Korotkov, A., Kostritsa, F., Landhuis, D., Lindmark, M., Lucero, E., Lyakh, D., Mandrà, S., McClean, J. R., McEwen, M., Megrant, A., Mi, X., Michielsen, K., Mohseni, M.,

Mutus, J., Naaman, O., Neeley, M., Neill, C., Niu, M. Y., Ostby, E., Petukhov, A., Platt, J. C., Quintana, C., Rieffel, E. G., Roushan, P., Rubin, N. C., Sank, D., Satzinger, K. J., Smelyanskiy, V., Sung, K. J., Trevithick, M. D., Vainsencher, A., Villalonga, B., White, T., Yao, Z. J., Yeh, P., Zalcman, A., and Martinis, H. N. J. M. (2019). Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510.

[Bacon et al., 2013] Bacon, D., Flammia, S. T., and Crosswhite, G. M. (2013). Adiabatic quantum transistors. *Physical Review X*, 3:021015:1–17.

[Bassman et al., 2021] Bassman, L., Beeumen, R. V., Younis, E., Smith, E., Iancu, C., and de Jong, W. A. (2021). Constant-depth circuits for dynamic simulations of materials on quantum computers.

[Bassman et al., 2020] Bassman, L., Liu, K., Krishnamoorthy, A., Linker, T., Geng, Y., Shebib, D., Fukushima, S., Shimojo, F., Kalia, R. K., Nakano, A., et al. (2020). Towards simulation of the dynamics of materials on quantum computers. *Physical Review B*, 101(18):184305.

[Bergholm et al., 2018] Bergholm, V., Izaac, J., Schuld, M., Gogolin, C., Alam, M. S., Ahmed, S., Arrazola, J. M., Blank, C., Delgado, A., Jahangiri, S., McKiernan, K., Meyer, J. J., Niu, Z., Száva, A., and Killoran, N. (2018). PennyLane: Automatic differentiation of hybrid quantum-classical computations.

[Bichsel et al., 2020] Bichsel, B., Baader, M., Gehr, T., and Vechev, M. (2020). Silq: A high-level quantum language with safe uncomputation and intuitive semantics. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 286–300, New York, New York, USA. Association for Computing Machinery.

[Bishop et al., 2017] Bishop, L., Bravyi, S., Cross, A., Gambetta, J., Smolin, J., and March (2017). Quantum volume.

[Bishop and Gambetta, 2019] Bishop, L. and Gambetta, J. (2019). Reduction and/or mitigation of crosstalk in quantum bit gates. US Patent App. 15/721,194.

[Boixo et al., 2012] Boixo, S., Albash, T., Spedalieri, F. M., Chancellor, N., and Lidar, D. A. (2012). Experimental signature of programmable quantum annealing. *arXiv:1212.1739*.

[Boixo et al., 2018] Boixo, S., Isakov, S., Smelyanskiy, V., Babbush, R., Ding, N., Jiang, Z., Bremner, M. J., Martinis, J., and Neven, H. (2018). Characterizing quantum supremacy in near-term devices. *Nature Physics*, 14:595–600.

[Boothby et al., 2021] Boothby, K., Enderud, C., Lanting, T., Molavi, R., Tsai, N., Volkmann, M. H., Altomare, F., Amin, M. H., Babcock, M., Berkley, A. J., Aznar, C. B., Boschnak, M., Christiani, H., Ejtemaee, S., Evert, B., Gullen, M., Hager, M., Harris, R., Hoskinson, E., Hilton, J. P., Jooya, K., Huang, A., Johnson, M. W., King, A. D., Ladizinsky, E., Li, R., MacDonald, A., Fernandez, T. M., Neufeld, R., Norouzpour, M., Oh, T., Ozfidan, I., Paddon, P., Perminov, I., Poulin-Lamarre, G., Prescott, T., Raymond, J., Reis, M., Rich, C., Roy, A., Esfahani, H. S., Sato, Y., Sheldan, B., Smirnov, A., Swenson, L. J., Whittaker, J., Yao, J., Yarovoy, A., and Bunyk, P. I. (2021). Architectural considerations in the design of a third-generation superconducting quantum annealing processor. arXiv:2108.02322v1 [quant-ph].

[Choi, 2011] Choi, V. (2011). Different adiabatic quantum optimization algorithms for the NP-complete exact cover problem. *Proceedings of the National Academy of Sciences*, 108(7).

[Cirac and Zoller, 1995] Cirac, J. I. and Zoller, P. (1995). Quantum computations with cold trapped ions. *Physical review letters*, 74(20):4091.

[Clarke and Wilhelm, 2008] Clarke, J. and Wilhelm, F. K. (2008). Superconducting quantum bits. *Nature*, 453(7198):1031.

[D-Wave Systems, Inc.,] D-Wave Systems, Inc. D-Wave Advantage system overview. `https://www.dwavesys.com/resources/white-paper/the-d-wave-advantage-system-an-overview/`. Accessed 20-May-2022.

[D-Wave Systems Inc.,] D-Wave Systems Inc. D-Wave Ocean software documentation.

[D-Wave Systems, Inc., a] D-Wave Systems, Inc. D-Wave Ocean software documentation, revision 6f16a2d3. `https://ocean.dwavesys.com/`. Accessed 2-Oct-2021.

[D-Wave Systems, Inc., b] D-Wave Systems, Inc. Operation and timing. Accessed 20-Jul-2022.

[Davis et al., 2019] Davis, M. G., Smith, E., Tudor, A., Sen, K., Siddiqi, I., and Iancu, C. (2019). Heuristics for quantum compiling with a continuous gate set.

[Dawson and Nielsen, 2005] Dawson, C. M. and Nielsen, M. A. (2005). The solovay-kitaev algorithm. *Quantum Information and Computation*.

[de Moura and Bjørner, 2008] de Moura, L. and Bjørner, N. (2008). Z3: An efficient SMT solver. In Ramakrishnan, C. R. and Rehof, J., editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340, Budapest, Hungary. Springer.

[De Vos and De Baerdemacker, 2015] De Vos, A. and De Baerdemacker, S. (2015). The block-zxz synthesis of an arbitrary quantum circuit. *Physical Review A*.

[Ding et al., 2020] Ding, Y., Gokhale, P., Lin, S., Rines, R., Propson, T., and Chong, F. T. (2020). Systematic crosstalk mitigation for superconducting qubits via frequency-aware compilation. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 201–214, Los Alamitos, CA, USA. IEEE Computer Society.

[Endres and Schindelin, 2003] Endres, D. M. and Schindelin, J. E. (2003). A new metric for probability distributions. *IEEE Transactions on Information Theory*, 49(7):1858–1860.

[Farhi et al., 2014] Farhi, E., Goldstone, J., and Gutmann, S. (2014). A quantum approximate optimization algorithm. Technical Report MIT-CTP/4610, Center for Theoretical Physics, Massachusetts Institute of Technology.

[Farhi et al., 2015] Farhi, E., Goldstone, J., and Gutmann, S. (2015). A quantum approximate optimization algorithm applied to a bounded occurrence constraint problem.

[Gabor et al., 2019] Gabor, T., Zielinski, S., Feld, S., Roch, C., Seidel, C., Neukart, F., Galter, I., Mauerer, W., and Linnhoff-Popien, C. (2019). Assessing solution quality of 3SAT on a quantum annealing platform.

[Gilchrist et al., 2005] Gilchrist, A., Langford, N. K., and Nielsen, M. A. (2005). Distance measures to compare real and ideal quantum processes. *Physical Review A*, 71(6).

[Gokhale et al., 2019] Gokhale, P., Ding, Y., Propson, T., Winkler, C., Leung, N., Shi, Y., Schuster, D. I., Hoffmann, H., and Chong, F. T. (2019). Partial compilation of variational algorithms for noisy intermediate-scale quantum machines. In *International Symposium on Microarchitecture*.

[Gokhale et al., 2020] Gokhale, P., Javadi-Abhari, A., Earnest, N., Shi, Y., and Chong, F. T. (2020). Optimized quantum compilation for near-term algorithms with openpulse. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 186–200, Los Alamitos, CA, USA. IEEE Computer Society.

[Green et al., 2013] Green, A. S., Lumsdaine, P. L., Ross, N. J., Selinger, P., and Valiron, B. (2013). Quipper: A scalable quantum programming language. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 333–342, New York, New York, USA. Association for Computing Machinery.

[Grover, 1996] Grover, L. K. (1996). A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, pages 212–219, New York, New York, USA. ACM.

[Hadfield et al., 2019] Hadfield, S., Wang, Z., O'Gorman, B., Rieffel, E. G., Venturelli, D., and Biswas, R. (2019). From the quantum approximate optimization algorithm to a quantum alternating operator ansatz. *Algorithms*, 12(2).

[IBM, 2020] IBM (2020). Fair-share queuing.

[IBM Quantum Services,] IBM Quantum Services. ibmq_brooklyn. Accessed 20-May-2022.

[ibm0,] ibm0. IBM Quantum Devices. `https://quantumexperience.ng.bluemix.net/qx/devices`. Accessed: 2018-05-16.

[JavadiAbhari et al., 2014] JavadiAbhari, A., Patil, S., Kudrow, D., Heckey, J., Lvov, A., Chong, F. T., and Martonosi, M. (2014). ScaffCC: A framework for compilation and analysis of quantum computing programs. In *Proceedings of the 11th ACM Conference on Computing Frontiers*, CF '14, pages 1:1–1:10, New York, NY, USA. ACM.

[JavadiAbhari et al., 2015] JavadiAbhari, A., Patil, S., Kudrow, D., Heckey, J., Lvov, A., Chong, F. T., and Martonosi, M. (2015). ScaffCC: Scalable compilation and analysis of quantum programs. *Parallel Computing*, 45:2–17.

[Johnson and Ben-Shach, 2022] Johnson, B. and Ben-Shach, G. (2022). Qiskit Runtime primitives make algorithm development easier than ever. Accessed 23-Aug-2022.

[Jouppi et al., 2018] Jouppi, N., Young, C., Patil, N., and Patterson, D. (2018). Motivation for and evaluation of the first tensor processing unit. *IEEE Micro*, 38(3):10–19.

[Khetawat et al., 2019] Khetawat, H., Atrey, A., Li, G., Mueller, F., and Pakin, S. (2019). Implementing NChooseK on IBM Q quantum computers. In Thomsen, M. K. and Soeken, M., editors, *Reversible Computing*, volume 11497 of *Lecture Notes in Computer Science*, pages 209–223. Springer.

[Kullback and Leibler, 1951] Kullback, S. and Leibler, R. A. (1951). On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79 – 86.

[Li et al., 2019] Li, G., Ding, Y., and Xie, Y. (2019). Tackling the qubit mapping problem for NISQ-era quantum devices. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, pages 1001–1014, New York, NY, USA. ACM.

[Li et al., 2020] Li, G., Ding, Y., and Xie, Y. (2020). Towards efficient superconducting quantum processor architecture design. In *Conference on Architectural Support for Programming Languages and Operating Systems*.

[Liu et al., 2020] Liu, J., Byrd, G. T., and Zhou, H. (2020). Quantum circuits for dynamic runtime assertions in quantum computation. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, page 1017–1030, New York, NY, USA. Association for Computing Machinery.

[Louw and McIntosh-Smith, 2021] Louw, T. R. and McIntosh-Smith, S. N. (2021). Using the Graphcore IPU for traditional HPC applications. In *3rd Workshop on Accelerated Machine Learning (AccML), HiPEAC 2021 Conference*, number 4986 in EasyChair preprint. European Network on High-performance Embedded Architecture and Compilation.

[Lucas, 2014] Lucas, A. (2014). Ising formulations of many NP problems. *Frontiers in Physics*, 2:5:1–5:15.

[Magesan et al., 2011] Magesan, E., Gambetta, J. M., and Emerson, J. (2011). Scalable and robust randomized benchmarking of quantum processes. *Physical Review Letters*, 106(18).

[McCaskey et al., 2020] McCaskey, A. J., Lyakh, D. I., Dumitrescu, E. F., Powers, S. S., and Humble, T. S. (2020). XACC: A system-level software infrastructure for heterogeneous quantum–classical computing. *Quantum Science and Technology*, 5(2):024002:1–23.

[McKay et al., 2018] McKay, D. C., Alexander, T., Bello, L., Biercuk, M. J., Bishop, L., Chen, J., Chow, J. M., Córcoles, A. D., Egger, D., Filipp, S., Gomez, J., Hush, M., Javadi-Abhari, A., Moreda, D., Nation, P., Paulovicks, B., Winston, E., Wood, C. J., Wootton, J., and Gambetta, J. M. (2018). Qiskit backend specifications for OpenQASM and OpenPulse experiments. *preprint arXiv:1809.03452*.

[Murali et al., 2019a] Murali, P., Baker, J. M., Abhari, A. J., Chong, F. T., and Martonosi, M. (2019a). Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers. *arXiv preprint arXiv:1901.11054*.

[Murali et al., 2019b] Murali, P., Linke, N. M., Martonosi, M., Abhari, A. J., Nguyen, N. H., and Alderete, C. H. (2019b). Full-stack, real-system quantum computer studies: Architectural comparisons and design insights. In *International Symposium on Computer Architecture*, pages 527–540.

[Murali et al., 2020] Murali, P., Mckay, D. C., Martonosi, M., and Javadi-Abhari, A. (2020). Software mitigation of crosstalk on noisy intermediate-scale quantum computers. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, page 1001–1016, New York, NY, USA. Association for Computing Machinery.

[Paykin et al., 2017] Paykin, J., Rand, R., and Zdancewic, S. (2017). $\mathcal{Q}$WIRE: A core language for quantum circuits. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, pages 846–858, New York, New York, USA. Association for Computing Machinery.

[Pednault et al., 2019] Pednault, E., Gunnels, J., Maslov, D., and Gambetta, J. (2019). On 'quantum supremacy'.

[Prabhakar et al., 2022] Prabhakar, R., Jairath, S., and Shin, J. L. (2022). SambaNova SN10 RDU: A 7nm dataflow architecture to accelerate software 2.0. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 65, pages 350–352.

[Preskill, 2018] Preskill, J. (2018). Quantum Computing in the NISQ era and beyond. *Quantum*, 2:79.

[Preskill, 2023] Preskill, J. (2023). Quantum computing 40 years later.

[Shi et al., 2019] Shi, Y., Leung, N., Gokhale, P., Rossi, Z., Schuster, D. I., Hoffmann, H., and Chong, F. T. (2019). Optimized compilation of aggregated instructions for realistic quantum computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, pages 1031–1044, New York, NY, USA. ACM.

[Siraichi et al., 2018] Siraichi, M. Y., Santos, V. F. d., Collange, S., and Pereira, F. M. Q. (2018). Qubit allocation. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, pages 113–125. ACM.

[Smith et al., 2017] Smith, R. S., Curtis, M. J., and Zeng, W. J. (2017). A practical quantum instruction set architecture. arXiv:1608.03355 [quant-ph].

[Steiger et al., 2018] Steiger, D. S., Häner, T., and Troyer, M. (2018). ProjectQ: An open source software framework for quantum computing. arXiv:1612.08091v2 [quant-ph].

[Svore et al., 2018] Svore, K., Geller, A., Troyer, M., Azariah, J., Granade, C., Heim, B., Kliuchnikov, V., Mykhailova, M., Paz, A., and Roetteler, M. (2018). Q#: Enabling scalable quantum computing and development with a high-level DSL. In *Proceedings of the Real World Domain Specific Languages Workshop 2018*, pages 7:1–10, New York, New York, USA. Association for Computing Machinery.

[Tannu and Qureshi, 2019a] Tannu, S. S. and Qureshi, M. K. (2019a). Ensemble of diverse mappings: Improving reliability of quantum computers by orchestrating dissimilar mistakes. In *International Symposium on Microarchitecture*, pages 253–265.

[Tannu and Qureshi, 2019b] Tannu, S. S. and Qureshi, M. K. (2019b). Mitigating measurement errors in quantum computers by exploiting state-dependent bias. In *International Symposium on Microarchitecture*, pages 279–290.

[Tannu and Qureshi, 2019c] Tannu, S. S. and Qureshi, M. K. (2019c). Not all qubits are created equal: A case for variability-aware policies for NISQ-era quantum computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*.

[Toffoli, 1980] Toffoli, T. (1980). Reversible computing. In de Bakker, J. and van Leeuwen, J., editors, *Automata, Languages and Programming*, pages 632–644, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Wille et al., 2016] Wille, R., Keszocze, O., Walter, M., Rohrs, P., Chattopadhyay, A., and Drechsler, R. (2016). Look-ahead schemes for nearest neighbor optimization of 1D and 2D quantum circuits. In *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 292–297. IEEE.

[Wilson et al., 2021] Wilson, E., Mueller, F., and Pakin, S. (2021). Mapping constraint problems onto quantum gate and annealing devices. In *2021 IEEE/ACM Second International Workshop on Quantum Computing Software (QCS)*, pages 110–117. IEEE.

[Wilson et al., 2020] Wilson, E., Singh, S., and Mueller, F. (2020). Just-in-time quantum circuit transpilation reduces noise. In *IEEE International Conference on Quantum Computing and Engineering (QCE)*.

[Wood,] Wood, C. J. Introducing Qiskit Aer: A high performance simulator framework for quantum circuits. https://medium.com/qiskit/qiskit-aer-d09d0fac7759.

[Wright et al., 2019] Wright, K., Beck, K. M., Debnath, S., Amini, J. M., Nam, Y., Grzesiak, N., Chen, J.-S., Pisenti, N. C., Chmielewski, M., Collins, C., Hudek, K. M., Mizrahi, J., Wong-Campos, J. D., Allen, S., Apisdorf, J., Solomon, P., Williams, M., Ducore, A. M., Blinov, A., Kreikemeier, S. M., Chaplin, V., Keesan, M., Monroe, C., and Kim, J. (2019). Benchmarking an 11-qubit quantum computer. *arXiv:1903.08181*.

[Younis,] Younis, E. Bqskit/qfactor.

[Younis et al., 2020] Younis, E., Sen, K., Yelick, K., and Iancu, C. (2020). Qfast: Quantum synthesis using a hierarchical continuous circuit space.

[Yuan et al., 2022] Yuan, C., McNally, C., and Carbin, M. (2022). Twist: Sound reasoning for purity and entanglement in quantum programs. *Proceedings of the ACM on Programming Languages*, 6(POPL):30:1–32.

[Zulehner et al., 2018] Zulehner, A., Paler, A., and Wille, R. (2018). An efficient methodology for mapping quantum circuits to the IBM QX architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.