Classical simulations of noisy commuting quantum computations

Iskren V. Vankov

s1322965

Honours project for the degree of BSc Computer Science and Physics

School of Informatics
University of Edinburgh
UK
April 3, 2017
Abstract

As research on building a scalable quantum computer is advancing, it is important to be able to check such a device’s correctness. One way to achieve verification is through classical simulations. Due to their exponential complexity, we can only perform smaller computations and then extrapolate results to the domain of quantum supremacy. In this project we extend the usual perfect simulations by also considering the architectural constraints and noise of a particular physical implementation. We aim to provide a general framework for constructing such experimentally realistic simulations. Two particular problems are described in order to exemplify how the more general methodology can be applied to specific scenarios. In our concrete examples we focus on fully commuting Instantaneous Quantum Polynomial-time (IQP) algorithms executed on the NQIT Q20:20 machine. The first example estimates the partition functions of random 2D complex-temperature Ising models in the circuit model of computation. The second example solves instances of IQP X-programs using measurement-based quantum computation (MBQC). We observe significantly different behaviour between the two in regards to their responses to noise.
Acknowledgments

I would like to thank my supervisors - Elham Kashefi and Petros Wallden - without whom this project would not have been possible. Thanks to Petros for all the discussions over this one year and also for the course on Quantum Computation he taught that gave me the needed theoretical basis. I am grateful to Elham for adopting me as an integral part of the research group, despite my simple undergraduate status, as well as for our weekly meetings that constantly redefined future directions. On the same note, thanks to the whole group for all the ambient knowledge I have acquired through our team meetings. In particular, I really want to thank Daniel Mills for the thorough introduction to IQP he provided me with, and also all the intermediate questions he helped us tackle.

I would also like to thank David Gosset and Patrick Rall both for the theoretical discussions and the software support related to the classical simulator.

Last but not least, I want to express my gratitude to Rad Ploshtakov for proofreading this report and his consequential suggestions for improvement.
Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(Iskren V. Vankov)
# Contents

1 Introduction  
1.1 Goals and a roadmap to achieving them .......................... 2  
1.2 Contributions list .................................................. 3  
1.3 Report structure .................................................... 3  

2 Preliminaries  
2.1 Dirac notation .......................................................... 5  
2.2 Postulates of Quantum Mechanics .................................. 5  
2.3 Entanglement ........................................................... 5  
2.4 The qubit ............................................................... 6  
2.5 Single-qubit gates ...................................................... 6  
2.6 Two-qubit gates ....................................................... 7  
2.7 Quantum teleportation ................................................ 8  
2.8 Quantum circuits ...................................................... 8  
2.9 Gottesman-Knill theorem ............................................. 9  
2.10 Measurement-based quantum computation (MBQC) ............... 10  

3 Instantaneous Quantum Polynomial-time (IQP)  
3.1 Definitions ............................................................ 13  
3.2 IQP X-program hypothesis test ..................................... 14  
3.3 Gap of binary polynomials .......................................... 14  
3.4 Ising model partition function ...................................... 15  

4 Clifford + T simulator  
4.1 Simulator operating principle ....................................... 19  
4.2 Piecing up subprocedures .......................................... 20  

5 NQIT Q20:20  
5.1 Architecture ........................................................... 23  
5.2 Noise ................................................................. 24  
5.2.1 Time-based ......................................................... 24  
5.2.2 Operation-based ................................................... 24  

6 General methodology  
6.1 Experiment steps ...................................................... 25  
6.2 Restricting to architectures ........................................ 25  
6.3 Incorporating noise ................................................... 26  

7 Experiments  
7.1 Ising model partition functions in the circuit model ............ 27  
7.1.1 Brute force vs. simulator ....................................... 27  
7.1.2 Perfect vs. noisy .................................................. 28  
7.1.3 Summary .......................................................... 31  
7.2 X-programs in MBQC .................................................. 31  
7.2.1 Brute force vs. simulator ....................................... 31  
7.2.2 Perfect vs. noisy .................................................. 33  
7.2.3 Summary .......................................................... 36  

8 Discussion  

9 References
1 Introduction

Governed by Moore’s law, our classical\footnote{Throughout this report, the word classical is used synonymously with non-quantum.} devices’ computational power has been exponentially growing throughout the past few decades. In recent years, however, this trend has been slowing down as transistors’ average size has become small enough for quantum effects to become a significant impediment \cite{1}. The field of quantum computation has arisen just in time to face this issue. Based on the same quantum effects that hinder our classical computers’ development, it turns them to our advantage instead. It is believed that quantum computation has the potential to increase our computational capabilities past the classical limit that we seem to have almost reached.

A number of significant theoretical applications have already been developed. Some famous examples are Shor’s polynomial-time number factorization algorithm \cite{2} and Grover’s $O(\sqrt{N})$ unstructured search algorithm of $N$ items \cite{3}. There also already exist practical applications that are commercially available. For instance, quantum key distribution offers unconditional security (i.e. such that does not depend on any computational assumptions, as is the classical cryptography case) for the procedure of generating a secret key between two parties \cite{4, 5}. In 2004, this protocol was utilized to secure a bank transfer for the first time and thus served as a proof-of-concept for quantum technology \cite{6}.

Building a scalable quantum computer, capable of running algorithms for big enough systems to achieve superiority over classical computers, has proven to be a difficult task that we are yet to solve. Different schemes are being researched for physically implementing such a device - ion traps \cite{7, 8, 9}, optical \cite{10}, superconducting \cite{11, 12, 13}, etc. The question of verification arises as those implementations become more advanced and powerful. We need to be able to establish whether they produce correct results. This leads us to the concept of classically simulating quantum computation. Generally exponentially hard, it is still possible for small enough calculations, as well as for some special cases. Verification through simulation sounds oxymoronic since the point is to verify computations in the domain of quantum supremacy where classical simulations are not possible by its own definition. A needed clarification then is that the goal is to consider smaller, simulable calculations in order to acquire a certain amount of understanding and then extrapolate our conclusions to the fully quantum domain.

Most often, simulations of perfect computation are considered in literature for the purposes of verification. In a realistic experimental setup however, noise corrupts the information and makes a scalable quantum computer hard to build. Its effects are often considered in two different contexts - isolated (closed) \cite{14} and non-isolated (open) \cite{15}. The latter type of noise is related to coupling with the environment and the consequential thermalization. Closed systems disregard this effect and instead focus on error caused by imperfections of the Hamiltonian and/or state preparation that induce chaotic behaviour to the many-body system hosting the computation. Physical implementations exhibit both types of noise and therefore they have been studied both theoretically and experimentally for different types of systems. In this project we wish to apply this knowledge onto existing perfect classical simulations in order to simulate noisy quantum computation \cite{16} and compare the two. Results from such an investigation can answer questions regarding the size and type of calculations which can be performed on a particular device to get reasonable results that noise would not distort too much. That can then be used to determine, for example, the amounts...
and types of error correction needed for different computations.

Despite focusing on several selected problems, this project’s main intention is to develop a general methodology for designing and executing noisy simulations. This makes it not only an excellent educational opportunity, but more importantly a toolbox to be used in future research. Motivation for choosing the topic comes from concrete ongoing work in our group on verification and noise. There already are plans to apply the outcome of this project to other current research - e.g. D. Mill’s IQP hypothesis tests investigation. In order for such adaptations to be as straightforward as possible, we have tried to keep both logic and actual code general and modular. The specific example problems described in this report serve the main purpose of illustrating how the developed general framework can be applied in practice. For this reason, instead of only discussing a few particular problem instances, we have considered different models of computation, problems, architectural restrictions, etc. We have thus provided the steps for randomization of instances and noise generation that can be adapted to other simulations.

1.1 Goals and a roadmap to achieving them

First, let us clearly define our general top-level goal. We want to develop the tools needed to simulate on a classical machine some quantum algorithms with the twist of incorporating the inevitable noise caused by the particular physical implementation of some quantum machine.

We employ a bottom-up approach in which we exemplify how our general methodology, discussed in section 6, can be applied to specific problems. There are a few levels of complexity to this task. First, simulating quantum computation classically is generally an exponentially hard problem. There exist, however, cleverer than bruteforce schemes that allow more efficient classical simulation for some particular quantum algorithms. We have chosen one such improved simulator to use in this project, as described in section 4. The particular choice was made not only because of its efficiency but also because I was already familiar with it, after having worked on its development as a software with one of its authors.

Another question that arises from our goal definition is which particular quantum algorithms to consider. Some types of quantum computation were indeed more suitable for our project than others. The main consideration in making this choice was to pick algorithms that are relatively straightforward to execute on the quantum machine we are dealing with. We require this because long and complex calculations would most probably completely fail when the external noise is added. Still, we want the problems solved to be non-trivial and computationally hard enough to make use of the expected quantum speedup. Therefore, we set out to pinpoint such algorithms that are easy to execute on the quantum device we have chosen while being hard to simulate classically. Discussion on making this choice, and details about it, are presented in section 3.

Finally, there is one more decision that our examples require us to make. This is the question of picking the particular physical implementation of some quantum machine. There already exist devices with some quantum capabilities - not necessarily universal. As already mentioned above, different approaches involve annealing, such as the D-Wave machine [17, 18], optics, ion traps, etc. None of those approaches has achieved universal and fully scalable quantum computation yet so it is unclear whether any of them can be considered as better than the others. The physical sources of error in those types of machines are different and, therefore, so are their noise models. Regarding our project, we had two main requirements for the device we choose to consider. Firstly, it had to
have enough computational power, i.e. number of qubits, in order to be able to stretch our classical simulations to the point of quantum supremacy. Secondly, because of the differences in error models, we wanted to pick a machine that already had a well developed theoretical framework for its intrinsic noise. Those requirements led us to choose to work on the NQIT Q20:20 engine, more about which is written in section 5.

We have so far addressed the issues of: What noise to apply? To which algorithm to apply it? How to simulate that classically? Having fixed those answers, we explore them for both models of quantum computation - circuit-based, and measurement-based (see section 2). Because of noise’s probabilistic nature, we employ a Monte-Carlo type of simulations [19], i.e. randomly generate problem instances and noise contributions for them a number of times. Then we aggregate results to determine the mean impact of noise on computations of the respective class.

1.2 Contributions list

- Researched possible types of quantum algorithms to use and picked IQP.
- Researched IQP and selected the X-program and random Ising model instances partition function problems.
- Researched NQIT’s architecture and noise model.
- Separately for both pairs - random Ising model instances in circuit model and X-programs in MBQC:
  - Wrote code to generate random instances of the problem.
  - Wrote a bruteforce simulation for intermediate verification.
  - Wrote simulation code using subprocedures from the Clifford+T simulator.
  - Wrote code to generate random instances of the problem, provided architectural constraints.
  - Provided a theoretical way to incorporate NQIT’s noise in the simulation.
  - Extended the simulation code to include the noise.
  - Compared aggregated results over many iterations of the perfect and noisy simulations.
- Generalized the used common methodology in order to make it adaptable to other problems.

1.3 Report structure

The report structure goes as follows. In section 2 we introduce all required preliminaries without assuming any prior knowledge of either quantum mechanics or quantum computation. Then we define the Instantaneous Quantum Polynomial-time (IQP) class of algorithms in section 3, and choose two particular members of it to consider. In section 4 we discuss the used classical simulator. The particular physical implementation of a quantum computer we consider, NQIT’s Q20:20 engine, is introduced in section 5. The general noisy simulation methodology we have developed is presented in section 6. Two examples of how to apply our toolbox to particular problems are provided in section 7. Finally, conclusions are summarized and possible future developments are outlined in section 8.
2 Preliminaries

In this section we provide a brief introduction into the field of quantum computation. Readers of relevant background may choose to skip it. No prior physics knowledge is expected but some linear algebra is highly recommended for proper understanding. Most of the section is based on two books - one on quantum mechanics [20] and one on quantum computation [21] - as well as on Petros Wallden’s lecture notes for the Quantum Computation course at the University of Edinburgh.

2.1 Dirac notation

Bra-ket (Dirac) notation is widely used in physics and for that reason has also been established as the standard in quantum computation. We write an abstract vector as a \( |\psi\rangle \), and its complex conjugate as a bra: \( \langle \psi | \). An inner product between two vectors, usually written as \( \langle \phi , \psi \rangle \) becomes the bra-ket \( \langle \phi | \psi \rangle \). Analogously, we can define the outer product of two vectors to be \( |\phi \rangle \langle \psi | \).

2.2 Postulates of Quantum Mechanics

**Postulate 1:** The state of any physical system is fully described by a vector \( |\psi(t)\rangle \) in a Hilbert space \( \mathcal{H} \), i.e. a complex vector space with defined inner product.

**Postulate 2:** Observables are represented by Hermitian (self-adjoint) operators on the Hilbert space \( \mathcal{H} \).

**Postulate 3:** Measurement of an observable \( \Omega \) on a state \( |\psi\rangle \) yields one of \( \Omega \)'s eigenvalues \( \omega \) with probability \( P(\omega) \propto |\langle \omega | \psi \rangle|^2 \) and changes the system’s state from \( |\psi\rangle \) to \( |\omega\rangle \).

**Postulate 4:** The time evolution of a closed system is a unitary transformation which obeys Schrödinger’s equation:

\[
\frac{d}{dt} |\psi(t)\rangle = \hat{H} |\psi(t)\rangle
\]

where \( \hbar \) is the reduced Planck constant and \( \hat{H} \) is the Hamiltonian operator. The latter is an operator whose eigenvectors \( |E\rangle \) are called stationary states and whose eigenvalues are their corresponding energy levels. A system in a stationary state remains in it indefinitely under time evolution (thus their name). Being a Hermitian operator, the Hamiltonian has a spectral decomposition:

\[
\hat{H} = \sum_E E |E\rangle \langle E|
\]

2.3 Entanglement

Another property of quantum mechanics, quoted as a postulate in some references, refers to combining Hilbert spaces of different systems into a single space. Given states \( |v\rangle \) and \( |w\rangle \) which belong to spaces \( V \) and \( W \) respectively, the joint state of the full system is described by \( |v\rangle \otimes |w\rangle \in V \otimes W \). The shorthand notation \( |vw\rangle \) is often used.
A state can then be defined as entangled if it cannot be written as a direct product of its component systems. A typical example is the so-called EPR or Bell pair:

$$|\Phi^+\rangle = \frac{|00\rangle + |11\rangle}{2}$$  \hspace{1cm} (3)

Such an entangled state illustrates the concept of non-locality, dubbed spooky action at a distance by Einstein. Based on the postulates stated above, a measurement on one of the particles will yield either 0 or 1, and will collapse the state to $|00\rangle$ or $|11\rangle$ respectively. The other particle’s state has then changed instantaneously, irrespective of the distance between them! However, it can be shown that this non-local influence cannot be used to send information faster than the speed of light which is consistent with general physics.

2.4 The qubit

The quantum analog of the classical bit is the qubit. It is a two-level system the states of which live in a two-dimensional Hilbert space. If we denote the basis states of the space with $|0\rangle$ and $|1\rangle$, then any qubit state is described by a linear combination of the basis vectors:

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle$$  \hspace{1cm} (4)

where normalization of the state requires

$$\langle\psi|\psi\rangle = 1 \implies |\alpha|^2 + |\beta|^2 = 1$$  \hspace{1cm} (5)

Equation 4 shows us the main difference between a bit and a qubit. The classical bit can take only one of two values - 0 or 1, while the quantum bit can be in any of the infinitely many states in its two-dimensional Hilbert space. Those linear combinations of basis states are referred to as superpositions.

Physically, a qubit can be implemented using different two-level systems that arise in nature. For instance, the abstract states $|0\rangle$ and $|1\rangle$ can be related to polarization of photons, up and down spins of an electron, two different energy levels of an electron in an ion, etc. Abstracting theoretical quantum computation as a mathematical framework allows us to reach general results that do not depend on the particular physical implementation. In some cases though, it is necessary to consider the experimental realization as it differs from the perfect theory. For instance, in this project we consider a particular physical implementation in section 5 in order to model the external noise.

Systems with more than two levels can also be considered. The bits in such higher-dimensional spaces are called qudits and can be defined similarly to qubits. We do not consider such systems in this project.

2.5 Single-qubit gates

Having defined the quantum equivalent of the classical bit, we can also define the analog to logical gates. They are unitary operators, represented by $2^n \times 2^n$ matrices for a $2^n$-dimensional Hilbert space defined by $n$ qubits. Therefore, single-qubit gates are simply $2 \times 2$ unitary matrices. Commonly used gates are the Pauli operators:
\[ X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \] (6)

Notice that \( X \) is the equivalent of the classical NOT gate since
\[ X |0\rangle = |1\rangle \quad X |1\rangle = |0\rangle \] (7)

Usually only \( X \) and \( Z \) are used, and \( Y \) can be constructed as \( Y = iXZ \).

The computational basis vectors \( |0\rangle \) and \( |1\rangle \) are actually the +1 and -1 eigenvectors of the Pauli-Z operator, respectively:
\[ Z |0\rangle = |0\rangle \quad Z |1\rangle = -|1\rangle \] (8)

We can define analogously the X-basis as:
\[ X |+\rangle = |+\rangle \quad X |-\rangle = -|-\rangle \] (9)

\[ |+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \quad |-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} \] (10)

A change of basis between \( X \) and \( Z \) bases can then be performed using the Hadamard gate:
\[ H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \] (11)

Finally, of particular interest are the phase gates
\[ R_\theta = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix} \] (12)

Two commonly used examples are the roots of the Pauli-Z gate - the \( S \)-gate, and the \( T \)-gate.
\[ S = \sqrt{Z} = R_{\pi/2} = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \quad T = \sqrt{S} = R_{\pi/4} = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix} \] (13)

The T-gate is also called, rather confusingly, a \( \pi/8 \) gate because it can be expressed as \( T = diag(e^{-i\pi/8}, e^{i\pi/8}) \) up to a global phase \( e^{i\pi/8} \) which can be discarded. In a circuit diagram, we denote a single-qubit gate \( U \) by \[ ] .

### 2.6 Two-qubit gates

Similarly, two-qubit gates are \( 4 \times 4 \) unitary matrices. An important type are the controlled gates for which an operation is applied to a target qubit depending on the value of a control qubit. Given a single-qubit gate \( U \), the corresponding controlled gate can be expressed as:
\[ \Lambda U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & U_{11} & U_{12} \\ 0 & 0 & U_{21} & U_{22} \end{pmatrix} \] (14)
Commonly used such gates are the Controlled-X ($\Lambda X$) gate that is analogous to the classical CNOT, and its Hadamard-rotated counterpart, Controlled-Z ($\Lambda Z$).

\[
\Lambda X = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \quad \Lambda Z = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}
\] (15)

In circuit diagrams, noting $\Lambda Z$'s symmetry with respect to control and target, they are denoted as:

\[
\begin{align*}
\Lambda X &= \\
\Lambda Z &= 
\end{align*}
\] (16)

Using the $\Lambda X$ gate, one can construct the highly useful SWAP operation which interchanges the states of two qubits. Its schematic symbol and equivalent circuit are defined as:

\[
\begin{align*}
\begin{array}{c}
\begin{tikzpicture}
\node (X) [circle, draw] {};
\node (Y) [circle, draw, below of=X] {};
\draw (X) -- (Y);
\end{tikzpicture}
\end{array}
&= \begin{array}{c}
\begin{tikzpicture}
\node (X) [circle, draw] {};
\node (Y) [circle, draw, below of=X] {};
\draw (X) -- (Y);
\end{tikzpicture}
\end{array}
\end{align*}
\] (17)

Having access to this gate makes it possible to move qubits around which is crucial when, for instance, two-qubit gates can only be applied to adjacent sites. Interestingly, this simple operation is the source of all non-classicality in the matchgate formalism [22].

### 2.7 Quantum teleportation

One application of entanglement is the so-called quantum teleportation. Given two parties, A and B, who share a Bell pair between them, the protocol allows A to send an unknown quantum state $|\psi\rangle$ to B via a Bell measurement. The two-qubit Bell basis for that measurement is defined by the four Bell states:

\[
|\Phi^\pm\rangle = \frac{|00\rangle \pm |11\rangle}{\sqrt{2}}, \quad |\Psi^\pm\rangle = \frac{|01\rangle \pm |10\rangle}{\sqrt{2}}
\] (18)

The algorithm works in 3 steps, as shown in figure 1:

- A applies the local operations $\Lambda X$ and $H \otimes I$ on the qubits in her possession, and then performs a Bell basis measurement on them.

- A sends to B the classical information of the measurement outcomes.

- B applies X and Z corrections according to the classical bits received from A. His qubit is now in the initial (unknown) state $|\psi\rangle$.

An important application of quantum teleportation to our project is discussed in section 5.1 where it is used to connect qubits from different ion traps.

### 2.8 Quantum circuits

Similarly to the classical case, we can construct logical circuits using the gates we have defined, and qubits as wires. There are a few differences, however, caused by the fact that unitary gates
are reversible. Circuits need to be acyclic, and the classical FANIN and FANOUT operations, i.e. joining and splitting wires, have no quantum equivalents. The set of allowed unitaries is uncountable but we can define finite universal gate sets for which any unitary operation can be approximated to arbitrary accuracy using only gates from the set. A particular universal gate set that we use in this project is \( \{ H, T, \Lambda X \} \).

Since we have a universal gate set of only single and two-qubit gates, this means that we can construct unitaries on arbitrary numbers of qubits only from them. One particular example that is used in this project is the Toffoli gate, or the control-control-not gate (\( \Lambda\Lambda X \)). An interesting fact is that the Toffoli gate on its own forms a universal gate set.

There is one final ingredient we need in order to be able to construct any quantum computation in the described circuit model - observable measurement. A computational basis measurement is denoted by \( \langle \psi \mid \rangle \). Measurements in other bases can be achieved by first applying unitary rotations to the target qubit.

\section*{2.9 Gottesman-Knill theorem}

Given an \( n \)-qubit circuit, we could try to simulate it using brute-force matrix multiplication. However, the states are \( 2^n \)-dimensional, and the operators are \( 2^n \times 2^n \) matrices which clearly makes the naive simulation exponentially hard. For certain circuits though, it is possible to achieve polynomial complexity.

The \textit{n-qubit Pauli group} is defined as all \( n \)-fold tensor products of the Pauli matrices and the identity, together with multiplicative factors \( \pm 1, \pm i \). For example, the single-qubit Pauli group is:

\[ G_1 = \{ \pm I, \pm iI, \pm X, \pm iX, \pm Y, \pm iY, \pm Z, \pm iZ \} \]  

Then we define the \textit{Clifford group} \( \mathcal{C} \) as the normalizer of the Pauli group, i.e. such a group that maps Paulis to Paulis:

\[ C \in \mathcal{C} \iff CPC^\dagger \in \mathcal{G} \quad \forall P \in \mathcal{G} \]  

Using those definitions we can quote the Gottesman-Knill theorem \[24\], that is essential for classical simulations of quantum computation, and therefore of great importance to our project:

\textbf{Gottesman-Knill theorem:} A quantum circuit containing only state preparations and measurements in the computational basis, and Clifford unitaries can be efficiently simulated classically.
2. PRELIMINARIES

The proof of this very important theorem relies on the *stabilizer formalism* but will be omitted from this report for the sake of brevity. The main point is that a Clifford computation can be performed in polynomial time and space by only keeping track of its generators.

The generators of the Clifford group are: \( C = \langle H, S, \Lambda X \rangle \). Therefore, we see that the *Clifford + T gate set is universal* because it contains the required gates \( \{ H, T, \Lambda X \} \) mentioned above. Combining this fact with the Gottesman-Knill theorem, we see that for any Clifford + T computation the exponential simulation complexity comes only from the T-gates, while all Cliffords are efficiently simulable. This fact becomes important in section 4 as it allows us to control the computational complexity of our simulations.

2.10 Measurement-based quantum computation (MBQC)

The circuit model is a straightforward extension of classical logic where we evolve a state through reversible unitary transformations, and measure the outcome at the end. It is not the only way to achieve universal quantum computation though. *Measurement-based quantum computation (MBQC)* \([25, 26]\) is another such framework. Its basic principle of operation is quite different from the circuit model. First, a big entangled state, called a *resource state* is prepared, and then the whole computation is done solely by measuring qubits in various bases. Such measurements are not unitary and as such are irreversible - the resource state is destroyed during execution. For this reason MBQC is also referred to as *one-way quantum computation*.

Initial entangling of the resource state is achieved by applying \( \Lambda Z \) between qubits in the \(| + \rangle \) state:

\[
\Lambda Z |+\rangle = \frac{1}{2} (|00\rangle + |01\rangle + |10\rangle - |11\rangle)
\]

(21)

This state is entangled by the definition given in section 2.3. Given a graph \( G = (V,E) \) with vertices \( V \) and edges \( E \), we can construct the initial resource state as:

\[
|G\rangle = \Pi_{(a,b)\in E} \Lambda Z_{a,b} |+\rangle^\otimes V
\]

(22)

Then it can be shown that a measurement on a single qubit in the \( \{ |+\rangle, |-\rangle \} \) basis where

\[
|\pm\rangle = \frac{1}{\sqrt{2}} \left( |0\rangle \pm e^{i\theta} |1\rangle \right)
\]

(23)

is equivalent to applying \( X^{s_j} J(-\theta) \) where \( s_j \in \{0,1\} \) is the measurement outcome, and \( J(\theta) \equiv HR_\theta \).

Using Euler’s rotation theorem, we can see that any unitary can be expressed as a product of three rotations and an extra Hadamard \( (H = J(0)) \):

\[
\exists \theta_1, \theta_2, \theta_3 : U = J(0)J(\theta_1)J(\theta_2)J(\theta_3) \quad \forall U \in U(2)
\]

(24)

Therefore the gate set \( \{ \Lambda Z, \{ J(\theta) \} \} \) is universal as well. This also proves MBQC’s universality because it consists only of generating the resource state using \( \Lambda Z \), and measurements in different bases which corresponds to applying \( J(\theta) \).

Because of the extra terms \( X^{s_j} \) in the link between measurement and applying \( J(\theta) \), it is necessary to apply corrections, based on the classical outcomes \( s_j \). Applying Paulis in between measurements to account for those unwanted terms would be inconsistent with MBQC’s concept of achieving
computation only through adaptation measurements, i.e., updating basis angles based on previous classical outcomes. Any MBQC computation (using a universal resource state) is defined by an ordered list of elements \( (i, \phi_i) \) that mean ”measure the \( i \)th qubit in the \( \{ \pm \phi_i \} \) basis.” The ordering needs to abide certain restrictions and most importantly be acyclic. Adaptive corrections are then achieved by updating the default set of angles to new values, according to previous measurement outcomes: \( \phi_i \to \phi'_i \) using equation 30. We present a brief sketch of its derivation, without going into excessive detail. More in-depth discussions are available in the references quoted at the beginning of this section.

Let us denote the given ordering of the measurements by a function \( f \) where \( f(i) \) is the qubit to be measured after the \( i \)th one. Using the fact that \( HX = ZH \), we see that further measurement on a qubit that already carries a term \( X^{s_1} \) will result in a \( Z^{s_2} \) error by canceling the two Hadamards from the two \( J(\theta) \) gates. This can be used to show that Pauli X and Z are the only two types of terms that we are aiming to correct. Furthermore, corrections for a qubit \( i \) may have only one X-term, coming from its immediate predecessor \( f^{-1}(i) \), and many Z-terms - one for each qubit \( j \) whose following qubit \( f(j) \) is a neighbour of \( i \). It can be shown that an X-correction is equivalent to flipping the sign of the angle \( \phi \to -\phi' \). Accounting for Z-corrections uses the fact that the initial resource state \( |G\rangle \) from equation 22 can be shown to be stabilized (i.e. \( K_i |G\rangle = |G\rangle \forall i \)) by the operators

\[
K_i = X_i \Pi_{j \in \mathcal{N}_G(i)} Z_j
\]

where \( \mathcal{N}_G(i) \) is the set of neighbours of \( i \) in the graph \( G \). Thus a Z-correction can be expressed as

\[
Z_i |G\rangle = Z_i K_{f(i)} |G\rangle = Z_i X_{f(i)} \Pi_{j \in \mathcal{N}_G(f(i))} Z_j |G\rangle = Z_i Z_i X_{f(i)} \Pi_{j \in \mathcal{N}_G(f(i)) \setminus \{ i \}} Z_j |G\rangle = X_{f(i)} \Pi_{j \in \mathcal{N}_G(f(i)) \setminus \{ i \}} Z_j |G\rangle
\]

Using this and the fact that the Z terms are equivalent to adding phases \( \pi \), the final expression for the adaptively updated measurement angles becomes:

\[
\phi'_i = (-1)^{s_{f^{-1}(i)}} \phi_i + \pi \sum_{j : j \in \mathcal{N}_G(f(j))} s_j
\]

2. PRELIMINARIES
2. PRELIMINARIES
3 Instantaneous Quantum Polynomial-time (IQP)

Following the roadmap from section 1.1, we need to select particular quantum algorithms to be considered in our classical simulations. In order to justify their choice, we first need to introduce the class of quantum computation they belong to - IQP.

3.1 Definitions

IQP (Instantaneous Quantum Polynomial-time): first introduced by Dan Shepherd [27, 28], it is a non-universal class of quantum computation that only allows diagonal gates. This restriction leads to the corollary that all gates commute and thus can, theoretically, be applied simultaneously. Despite being non-universal, IQP has been shown to still be hard to simulate classically [29]. This complexity result relies on the common assumption of the conjectured structure of the polynomial hierarchy (PH) [30]. The latter is a hierarchy of complexity classes, which are recursively generated by adding oracles to previous levels. The more precise statement about IQP’s complexity then is that if there existed an efficient classical algorithm to sample from the IQP output distribution up to a certain multiplicative error, the PH would collapse to its third level. While not as implausible as P = NP, which is a collapse to the zeroth level, it is still considered highly improbable. The non-classicality comes from \( R_\theta \) gates that are diagonal and thus part of IQP. One particular example that will be important in our later considerations is the T-gate 

\[
T = |0\rangle \langle 0| + e^{i\pi/4} |1\rangle \langle 1|
\]

One can recognize the individual terms

\[
exp \left( i\theta_p \bigotimes_{j:p_j=1} X_j \right)
\]

as corresponding to Hamiltonians that act on different subsets of all qubits with the Pauli X operator on each with actions \( \theta_p \). Every IQP circuit, i.e. composed of commuting diagonal gates, can be described as an X-program. Since by definition all allowed gates commute with each other, the computation lacks any temporal structure. Ordering of the gates does not matter, and the X-program notation shows this directly by defining the whole computation as a product of commuting Hamiltonians. A corollary of this definition, valid for any IQP circuit, and that will be used later, is the possible Hadamard change of basis for the full circuit, i.e.:
3. INSTANTANEOUS QUANTUM POLYNOMIAL-TIME (IQP)

\[ P(X = x) = \left| \left\langle x | \exp \left( \sum_{p} i \theta_p \bigotimes_{j: \mathbf{p}_j = 1} X_j \right) \right| 0 \right\rangle \otimes \left| 0 \right\rangle \right|^2 \]  
\[ = \left| \left\langle x | H \otimes \exp \left( \sum_{p} i \theta_p \bigotimes_{j: \mathbf{p}_j = 1} Z_j \right) H \right| 0 \rangle \otimes \left| 0 \right\rangle \right|^2 \]  
\[ = \left| \left\langle y | \exp \left( \sum_{p} i \theta_p \bigotimes_{j: \mathbf{p}_j = 1} Z_j \right) \right| + \right\rangle \otimes \left| + \right\rangle \right|^2 \]

Having introduced IQP, now we can justify our choice to pick it as a good target class for our simulations. As described in its definition, it addresses both considerations we had for deciding on particular algorithms. Firstly, it is expected to be relatively simple to execute on a quantum device. This follows from its property that all unitaries have to commute with each other. Therefore, as its name suggests, a computation can happen in one timestep, at least theoretically, negating noise contributions such as depolarisation and dephasing. More on error sources can be found in section 5. Secondly, it is a true quantum computation since it is hard to simulate classically. Those two properties make IQP a good choice for our goals. In the next sections we discuss our choice of particular algorithms within IQP.

3.2 IQP X-program hypothesis test

One particular problem that can be solved in IQP was presented in one of the initial papers [28]. If we set all angles to be the same, \( \theta_p = const = \pi/8 \), we can represent any particular X-program by writing an \( \mathcal{O}(n) \times n \) matrix \( X \), each line of which is one of the binary vectors \( \mathbf{p} \). Using this constant angle assumption, we can still generate X-programs that are IQP-hard. They can be used as a means for a server to prove it has IQP oracle capabilities in such a way that the client can verify that without having any quantum computational power. The authors have posted on-line a $25 challenge [31] that generate such verifiable IQP-hard X-programs. For the brevity of this report, we will not describe the whole protocol, which can be found in [28]. The main idea behind it is that they generate big X-programs (\( n \sim 500 \)) by randomly extending and reordering binary matroids in such a way that the output distribution is biased in the direction of some client-input secret vector \( \mathbf{s} \in \mathbb{F}_2^n \). Then output from a true IQP oracle is expected to be orthogonal to \( \mathbf{s} \) more often compared to output of a classical approximation (86% to 75%) which is used by the client to determine whether the server is indeed capable of IQP.

This hypothesis test is a particular way of verification of quantum devices. We consider it in our MBQC simulations in section 7.2.

3.3 Gap of binary polynomials

Further work by the same people shows two particular problems that can be solved in IQP [32]. One of them is about finding the gap of degree-3 polynomials over \( \mathbb{F}_2 \). This is a mathematical problem which can be shown to have applications in evaluating boolean expressions in computer science [33]. Let us consider a boolean function \( f : \{0,1\}^n \rightarrow \{0,1\} \) which is also a polynomial of degree 3, i.e.
3. INSTANTANEOUS QUANTUM POLYNOMIAL-TIME (IQP)

\[ f(x) = \sum_{ijk} \alpha_{ijk} x_i x_j x_k + \sum_{ij} \beta_{ij} x_i x_j + \sum_i \gamma_i x_i \pmod{2} \]  

We can define the gap of this function as:

\[ \text{gap}(f) \equiv \left| \{ x : f(x) = 0 \} \right| - \left| \{ x : f(x) = 1 \} \right| \]

Then, an IQP circuit can estimate this gap for a particular instance of this problem, defined by \( \alpha_{ijk}, \beta_{ij}, \gamma_i \in \{0, 1\} \) by using

\[ \text{gap}(f) = 2^n \langle 0 | \otimes^n C_f | 0 \rangle \otimes^n \]

where \( C_f \) is the circuit generated by transferring every \( \gamma_i x_i \) term to a \( Z \) gate, every \( \beta_{ij} x_i x_j \) term to a \( \Lambda Z \) gate, and every \( \alpha_{ijk} x_i x_j x_k \) term to a \( \Lambda\Lambda Z \) gate. Notice that both \( Z \) and \( \Lambda Z \) are Cliffords, and therefore efficiently simulable, whereas the Toffoli (after a Hadamard rotation) \( \Lambda\Lambda Z \) requires using a gadget that uses 4 T-gates - cf. figure 1 of [34].

This is a good example of an application of IQP for a meaningful problem from another field. However, it does not comply with one of our requirements for choosing an algorithm, listed in 1.1. The Toffoli gates needed for its implementation are three-qubit operations that are harder to physically implement in a particular device. Therefore we have chosen not to focus on this algorithm. We still include it in the report because it can be used for further research on the topic. The same methodology we have used for the algorithms we chose to consider can be applied to this problem too, and this can be useful when considering quantum devices with straightforward Toffoli implementations.

3.4 Ising model partition function

The other IQP problem described in [32] is about estimating the complex-temperature partition function for random instances of the 2D Ising model. It is a commonly used method in statistical physics to describe ferromagnetism. The 2D lattice we consider is one of the simplest models to exhibit a phase transition. The concept of a partition function has a central role in statistical physics (not only for Ising models). It has the meaning of a normalization constant for the microstates’ probabilities, and its derivatives can be used to determine the thermodynamic quantities of the system. Generalizing temperatures to complex values is an example of analytic continuation and has been used to gain deeper insight into the problem [35]. Without magnetic fields (i.e. single-qubit rotations), the problem is exactly solvable. However, it has been shown in [36] that adding random magnetic fields makes the problem computationally hard under similar assumptions to the general IQP case.

Consider the partition function \( Z(\omega) \) defined as:

\[ Z(\omega) = \sum_{z \in \{\pm 1\}^n} \omega^{\sum_{i<j} w_{ij} z_i z_j + \sum_{k=1}^n v_k z_k} \]

If we set the complex temperature to a constant, \( \omega = e^{i\pi/8} \), then any particular instance of the Ising model is fully defined by specifying the edge weights \( w_{ij} \) and vertex weights \( v_k \) of an n-vertex
3. **INSTANTANEOUS QUANTUM POLYNOMIAL-TIME (IQP)**

|0⟩ \( H \) \( Z^{\frac{3}{2}} \) \( T \) \( H \)

|0⟩ \( H \) \( Z \) \( T^7 \) \( H \)

|0⟩ \( H \) \( Z^{\frac{1}{2}} \) \( Z^{\frac{3}{2}} \) \( Z \) \( T \) \( H \)

Figure 2: Example circuit solving the random Ising model instance defined in equation 40. The fact shown in equation 32 has been used here in order to use roots of \( \Lambda Z \) instead of \( \Lambda X \).

A particular example is shown in figure 2, generated by the following graph weights:

\[
W = \begin{pmatrix}
0 & 3 & 2 & 2 \\
0 & 0 & 2 & 3 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0
\end{pmatrix}, \quad V = \begin{pmatrix}
4 \\
0 \\
7 \\
1
\end{pmatrix}
\] (40)

As mentioned earlier, any Clifford gate is efficiently simulable, and only T-gates are exponentially hard. Let us then roughly estimate the classical hardness by considering how the total T-gate count is affected by different values for the weights. Using

\[
T^{v_k=4} = Z
\] (41)

and

\[
T^{v_k=2} = S
\] (42)

we see that any even number of powers of T combine into a Clifford gate, meaning that vertex weights \( v_k \) increase the total T-gate count by \( f(v_k) \) being:

\[
f(v_k) = \begin{cases} 
0 & \text{if } v_k \text{ even,} \\
1 & \text{if } v_k \text{ odd.}
\end{cases}
\]
Regarding the edge weights, even powers are Cliffords, i.e.

\[ \Lambda S^{w_{ij}=0} = I \]  \hspace{1cm} (43)

and

\[ \Lambda S^{w_{ij}=2} = \Lambda Z \]  \hspace{1cm} (44)

whereas odd powers lead to \( \Lambda S \) gates that are non-Clifford:

\[ \Lambda S^{w_{ij}=1} = \Lambda S \]  \hspace{1cm} (45)

and

\[ \Lambda S^{w_{ij}=3} = \Lambda Z \Lambda S \]  \hspace{1cm} (46)

Since \( \Lambda S \) is not part of our simulator’s gateset, we need to substitute it with an equivalent gadget. Using a paper on representing roots of Paulis [37], we devised the gadget shown below that uses 3 T-gates.

\[ \begin{array}{c}
\text{T} \\
\text{T} \\
\text{T}^+ \\
\end{array} = \begin{array}{c}
\text{S} \\
\end{array} \]  \hspace{1cm} (47)

Therefore edge weights \( w_{ij} \) increase the total T-gate count by \( g(w_{ij}) \) being:

\[ g(w_{ij}) = \begin{cases} 
0 & \text{if } w_{ij} \text{ even,} \\
3 & \text{if } w_{ij} \text{ odd.} 
\end{cases} \]

Combining those observations, the T-gate count of a whole circuit, defined fully by \( \{v_k\} \) and \( \{w_{ij}\} \), is

\[ t = \sum_{ijk} (f(v_k) + g(w_{ij})) = (\# \text{ odd } v_k \text{s}) + 3(\# \text{ odd } w_{ij} \text{s}) \]  \hspace{1cm} (48)

Therefore we can independently vary the two sources of non-classicality in the simulation - number of qubits, and T-gate count - by biasing the random circuit generator towards even or odd weights, as desired.
3. INSTANTANEOUS QUANTUM POLYNOMIAL-TIME (IQP)
4 Clifford + T simulator

Going back to our roadmap, we need to pick a classical simulator to base our code on. As discussed in the introduction section, we have chosen to work with the improved Clifford + T classical simulator that was described by Bravyi and Gosset [38]. It is appropriate for the Ising model instances problem because, as shown in section 3.4, it involves circuits made up of Cliffords and the T gate, after applying the appropriate gadgets. Furthermore, we showed that we can bias the random circuit generation towards arbitrarily low T-gate counts that are related to input weights’ parity, as seen in equation 48. This can allow for our circuits to be truly Clifford-dominated, while still being classically hard to simulate, which is the setting this simulator was originally developed for, as its paper name suggests. Analogously, we have also shown in section 7.2 for all of the above to be also true for the other problem we consider - the X-programs from section 3.2 in the MBQC picture. Through this second example we have demonstrated the extended MBQC capabilities of the originally circuit-based simulator. Therefore its choice is justified for both problems within this project’s scope.

The other reason for using this particular simulator is that I was already familiar with it, after working on its development with one of its authors, David Gosset. Together with another undergraduate at Caltech, Patrick Rall, we developed more optimized Python + C code based on the same paper [39]. Despite being slower, the original MATLAB codebase is used in this dissertation because we have found cases in which the Python + C code is not yet stable.

4.1 Simulator operating principle

The Gottesman-Knill theorem, defined in section 2.9 tells us that we can efficiently simulate Clifford circuits. Adding a T-gate to the gate set makes it universal at the cost of exponential complexity for classical simulation. The power of the simulator from [38] is in its reduction of the exponential complexity dependence to $\text{poly}(n,m) + 2^{t} \cdot \text{poly}$ for computing the probability of a given output for $n$ qubits, $m$ gates, and $t$ T-gates. The algorithm description can be found in [38] and will not be presented in full here in order to keep the report succinct. We provide a brief overview though, which is needed to understand our decisions regarding code architecture.

The algorithm’s speedup is achieved in 3 steps. First, all T-gates are replaced by the gadget from figure 3. That way the non-classicality from $t$ T-gates has been moved to the magic ancillas $|A\rangle \otimes t$. The second step approximates this initial magic state as a linear combination of $\chi \sim 2^{0.23t} \ll 2^n$ stabilizer states (simultaneous $+1$ eigenstates of the n-qubit Clifford group). Then the circuit, gadgetized to contain only Cliffords, is efficiently applied to each of those states. The final step approximates the norm of this linear combination. Then the final probability of a particular outcome is the ratio of two such norms (equations 26 and 27 from [38]):

$$P_{\text{out}}(x) = \frac{\langle 0^n \otimes \psi | V_{y}^{\dagger} (I_{n} \otimes |y\rangle \langle y|) V_{y} | 0^n \otimes \psi \rangle}{\langle 0^n \otimes \psi | V_{y}^{\dagger} (I_{n} \otimes |y\rangle \langle y|) V_{y} | 0^n \otimes \psi \rangle}$$

$$= \frac{2^{-u} \langle \psi | \Pi_{G} | \psi \rangle}{2^{-v} \langle \psi | \Pi_{H} | \psi \rangle}$$

where $|\psi\rangle$ is the linear superposition of the $\chi$ stabilizer states, $y$ is a bit string of length $t$ chosen at
random as outputs of the gadget measurements, \( V \) is our circuit, \( u \) and \( v \) are integers, and

\[
\Pi_Q = 2^{-|Q|} \sum_{P \in Q} P, \quad Q = G, H \tag{51}
\]

are projectors onto the codespace spanned by the states stabilized by \( Q \).

### 4.2 Piecing up subprocedures

The MATLAB code from the original paper only defines the subprocedures discussed in it and its appendices, rather than being a full input-to-output simulator. Therefore we had to write code to adapt the provided routines to our problem. A significant difference between the example in the paper and our needs is that the authors have simulated a hidden shift algorithm where all qubits are sampled independently, whereas we are interested in exactly one output amplitude - of the \( |0\rangle \otimes^n \) state - but without the independence assumptions. The simulator has this capability but some code had to be written to get from a circuit definition to the probability output in equation 49.

The defined gate set for the simulator’s routines involves: Paulis, Toffoli, Hadamard, \( T, T^\dagger \), \( S, S^\dagger \), \( \Lambda X \). Particular functions we have used, whose names can be recognized from the explanations in section 4.1, are:

- **GadgetizeCircuit** which converts \( T \)-gates to the gadgets of figure 3.
- **ChooseSubspace** which randomly tries \( \text{numtries} \) different \( k \)-dimensional spaces \( L \subseteq \mathbb{F}_2^k \) and outputs the one whose state \( |L\rangle \) has the highest fidelity with \( |H^\otimes t\rangle \) (equivalent to \( |A^\otimes t\rangle \) up to some Cliffords). Refer to section 5 of [38] for a more in-depth explanation of this procedure.
- **ComputeStabilizerGroup** which computes \( G, H \) and the integers \( u, v \) from equation 49.
- **NormEstimate** which computes the norms \( \langle \psi | \Pi_{G, H} | \psi \rangle \) from equation 49 by generating \( \text{samp} \) random stabilizer states to use in the approximate decomposition.

The simulation procedure then, using the provided routines follows those pseudocode steps:
Algorithm 1 GetZeroOutputProbability(samp, numtries, fidelityBound)

1: Populate an array of all unitaries in the desired circuit
2: Gadgetize gates not in the simulator’s gateset (e.g. $\Lambda S$ using equation 47)
3: GadgetizeCircuit()
4: Initialize $k = t/2$
5: while $F(|L\rangle, |H^{\otimes t}\rangle) < $ fidelityBound do
6: \hspace{1em} $F, |L\rangle \leftarrow$ ChooseSubspace($k, t, \text{numtries}$)
7: \hspace{1em} $k++$
8: end while
9: $G \leftarrow$ ComputeStabilizerGroup()
10: prob0n $\leftarrow$ NormEstimate($G$) (numerator in equation 49)
11: $H \leftarrow$ ComputeStabilizerGroup()
12: sumProb $\leftarrow$ NormEstimate($H$) (denominator in equation 49)
13: return prob0n / sumProb
4. CLIFFORD + T SIMULATOR
Following our roadmap from section 1.1 further, we need to make one final decision before we can move onto the experiments. This is the choice of a particular physical implementation of a quantum computer whose architecture and noise to adopt in our simulations. We have selected NQIT’s Q20:20 engine (also referred to just as NQIT below) for this purpose. The introduction they provide on their website \[40\] goes as follows:

The Networked Quantum Information Technologies Hub (NQIT) is the largest of the four Hubs in the UK National Quantum Technology Programme, a £270 million investment by the UK government to establish a quantum technology industry in the UK. We are working towards building a quantum computer demonstrator, the Q20:20 engine, which demonstrates a networked, hybrid light-matter approach to quantum information processing.

We have chosen this quantum device because it fits in both our requirements, set in 1.1. Firstly, it has a high enough number of qubits to allow computations beyond classical brute-force capabilities. Secondly, a thorough theoretical description of its noise, derived from a physics point of view, has been developed in \[41\]. Furthermore, the University of Edinburgh itself is an official partner of the project with one of my supervisors, Elham Kashefi, being the Quantum Hub’s Director of Applications. Members of our research group are currently working on its development, which allowed a number of useful discussions on the way. Our work on NQIT is relevant to ongoing research since they have planned to provide funds for the creation of a simulator, specific to their architecture and noise. Our example problems in section 7 serve this exact purpose but restricted to IQP algorithms. Using the general methodology from section 6, one could build on top of this project to create the desired universal NQIT simulator.

5.1 Architecture

The Q20:20 engine’s proposed structure involves \( N = 20 \) ion traps (\textit{cells}) with \( K = 20 \) ions in each. Every ion can be thought of as one physical qubit, as described in section 2.4. Cells are ordered on a 2D grid which means that every trap communicates with at most \( D = 4 \) others, assuming only nearest-neighbour interactions are allowed between cells. Traps are connected with entanglement links between specifically dedicated \textit{linking qubits}. Those links are realized through the procedure of \textit{entanglement distillation} \[42, 8\] which generates high-fidelity shared Bell pairs (equation 3). This process can be repeated a number of times with different numbers of \textit{link distillation qubits}, leading to different fidelities of the link. This means that some physical qubits need to be dedicated to distillation and linking, so not all \( K \) ions within a trap can be used as \textit{logical qubits}. We denote the number of logical qubits per cell we use with \( K' \). The number \( K' \) can be further lowered if error-correction codes are applied. Since the project is still underway, the system parameters \( N \), \( K \), \( K' \), \( D \), and others (to be discussed), are not set in stone. For this reason we let them vary in our simulation toolbox. Single and two-qubit gates within a trap take place in special \textit{gate zones}. Two-qubit gates between ion traps are applied by first \textit{teleporting} (cf. section 2.7) the qubits into the same cell, which consumes the shared Bell pair.
5.2 Noise

A thorough description of the noise model, justified through its physical origin, is available in section 2.5 of [41]. Here we present a brief summary of all types of noise and their parameter values. We split them into two types - time-based and operation-based. The former are errors that occur randomly in time on each physical qubit independently with some constant rate. Therefore we naturally model them as Poisson processes, as described in section 6.3. Operation-based noise, on the other hand, occurs when an operator is being applied, and is only applied on the qubits being acted upon with some probability.

5.2.1 Time-based

Depolarisation Caused by scattering of amplitudes of the electron’s wave-function between different energy levels of the ion. Modelled by a rate of $\approx 9 \times 10^{-4}$ per second for a random single-qubit Pauli on each qubit.

Dephasing Entanglement reduction that destroys data not stored in the standard basis. Can be modelled by a rate of $\approx (7.2 \pm 1.4) \times 10^{-3}$ per second for a $Z$ gate on each qubit.

Particular operation times In order to apply the noise channels of this section we need to know the time different parts of the computation will take.

- Preparation - 1 – 1.5ms
- Measurement - 2 – 2.5ms
- Any single or two-qubit operation within a trap (including moving the qubits to the gate zone) - 0.5ms
- Linking - for the case of 10 distillation qubits - 1 – 2s

5.2.2 Operation-based

Preparation Error probability in preparing a state. Modelled by a rate of $\approx 2 \times 10^{-4}$ for a Pauli $X$.

Measurement Similarly to preparation, measurement is noisy, too. It is estimated to have a rate of $\approx 5 \times 10^{-4}$ to measure wrongly any qubit, which corresponds to an $X$ gate in computational basis.

Single-qubit gates Random Pauli operator applied in addition to the single-qubit gate with probability $\approx (1.5 \pm 0.45) \times 10^{-6}$.

Two-qubit gates Modelled by independent single-qubit random Pauli errors on both qubits, each with probability $\approx (5.5 \pm 3.5) \times 10^{-4}$ and a further two-qubit error $Z \otimes Z$ with probability $\approx 6 \times 10^{-5}$.

Linking operations Depending on the amount of entanglement distillation used, this error will vary. The full dependence between those two is not provided but we know that the error is approximately the same as in the regular two-qubit noise if 10-qubit distillation is used.
6 General methodology

Having researched and presented all needed information about IQP, NQIT architecture and noise, and the Clifford + T simulator, we can move onto the essence of our project. In this section we describe how we connect all of the above into one framework. This toolbox contains all the necessary pieces to allow us to run different experiments, as defined in our initial goals in section 1.1.

6.1 Experiment steps

We refer to an experiment as the combination of preliminary theoretical work, e.g. devising ways to restrict a particular computation to the architecture, and the subsequent simulation code that implements the theoretical decision taken. Given a problem we want to consider, preparing and running an experiment in the general case is structured as:

1. Experiment part 1
   - Write code to generate random instances of the problem.
   - Write a bruteforce simulation.
   - Write code that adapts the simulator to solve those instances.
   - Compare bruteforce and aggregated simulator answers over a number of iterations to verify that the simulator works properly, and establish its average accuracy.

2. Experiment part 2
   - Write code to generate random instances of the problem, restricted to the quantum device’s architecture.
   - Devise a scheme to incorporate the noise for the particular problem and model of computation. Write code to randomly add noise to a restricted instance of the problem.
   - Compare perfect and noisy aggregated answers over a number of iterations to estimate the noise’s influence.

6.2 Restricting to architectures

In our case, restricting random instances to NQIT’s architecture refers to keeping the number of connections between different ion traps minimum. The machine itself is universal, meaning that it can theoretically run any problem instance without alterations. However, we impose the restriction in order to minimize the influence of entanglement distillation between cells, which is by far the slowest operation and thus the biggest noise contributor. Essentially, we pick only such random instances of the problems that fit naturally in NQIT’s networked architecture, i.e. when computation is mainly performed within the cells, and not between them. There is no unique way to do this, and different choices can lead to very different subsets of possible instances. The main criterion we have considered when deciding on a particular set of constraints was preserving the expected computational hardness of the problem. This rather abstract and general procedure will be made clearer through the examples in section 7. A detailed description of low-level circuit compilation for ion trap architectures can be found in [43].
6. GENERAL METHODOLOGY

6.3 Incorporating noise

The noise description in section 5.2 is specific to NQIT. However, its structure is general and other quantum devices will likely have similar "specification sheets" with different values. For example, noise can always be divided into time-based and operation-based, dephasing always destroys entanglement at some rate, etc. This generality allows our toolbox to be relatively easily adaptable to other quantum computers in future research.

The noise incorporation procedure, given a constructed perfect set of unitaries, goes as follows. All operations (qubits’ initial preparation, all gates from the given set, and all measurements) are considered independently. Noise gates are inserted at an operation’s position in the circuit randomly, with type and probability according to the provided rates (section 5.2). Also independently for each of those operations, a nested loop iterates over all qubits in the whole system and randomly applies the two time-based errors with probabilities according to Poisson processes with the rates from section 5.2 and the execution time of the current operation. Therefore, each operation can induce any type of operation-based error on its (one or two) application qubits, and also any type of time-based error on any qubit in the system. The concept behind the latter is that while some operation is being performed on a subset of the qubits, all of them independently dephase, depolarize, etc.

We assign the following meaning to the word *random* when referring to adding noise gates. Randomly adding operation-based noise gate is achieved by picking a number \( x \in (0, 1) \) at random and comparing it to the corresponding error rate \( \lambda \), listed in section 5.2. If \( x < \lambda \), the appropriate noise gate is added at the position of the original gate, otherwise the operation remains perfect. Randomly adding time-based noise for an operations acts in a similar way. First we calculate the execution time \( \tau \) needed for the current operation by considering the times given in section 5.2. It is important to be careful whether the operation involves using an entangled link between different traps, as that increases the time needed significantly. We then iterate over all qubits in the system and for each generate a random number \( x \in (0, 1) \). For each of those \( x \), we add an appropriate noise gate (again, their type is defined in section 5.2) to the corresponding qubit in the iteration according to a Poisson process with rate \( \lambda \). That is, a noise gate is added if \( x < e^{-\tau \lambda} \). In the remainder of this report, the word *random* refers to the logic presented in this paragraph, when referring to adding noise gates.
7 Experiments

In section 6 we discussed the steps needed for a noisy simulation of some problem. Now we present their application to two particular problems. The first one we consider is the random Ising model instances’ partition functions from section 3.4 in the circuit model of computation. The second one is about X-programs from section 3.1 in MBQC (section 2.10). For both problems we utilize our toolbox, both theoretically and code-wise, by following the steps provided. In future research, one can follow a similar route as those examples we provide in order to obtain results for any computation / architecture / noise / model of computation, as desired.

7.1 Ising model partition functions in the circuit model

We consider the problem of estimating the partition functions of random complex-temperature Ising model instances, defined in section 3.4, in the circuit model of computation. As explained in more detail in the aforementioned section, each instance of the problem is fully defined by an upper triangular matrix $W$ with integer elements $w_{ij} \in \{0, \ldots, 3\}$ and a vector $V$ with integer entries $v_k \in \{0, \ldots, 7\}$. $W$ and $V$ can be thought of as the edge and vertex weights of a graph. Considering this problem in the circuit model is then natural because every non-zero weight straightforwardly translates to a gate in the equivalent circuit $C_I$:

$$w_{ij} \rightarrow \Lambda S^{w_{ij}} \quad v_k \rightarrow T^{v_k}$$

(52)

7.1.1 Bruteforce vs. simulator

Following the steps from 6.1, our first goal is to investigate how accurately our simulator solves the problem compared to an exact bruteforce calculation. Naturally, we can only perform this comparison for small instances because of the steep exponential complexity dependence of the bruteforce method, explained in section 2.9.

In the first step we write a generator for random instances. In this straightforward procedure we initialize the $W$ matrix and $V$ vector with zero values. Then we iterate over their elements and randomly change them to integers (from their respective allowed value sets) if a randomly generated number is less than some threshold. By varying the $W$ and $V$ threshold values then we can achieve lower or higher T-gate counts on average. Given an instance, the bruteforce simulator iterates over $W$ and $V$ and multiplies matrices according to equation 52. Code for the generation and exact solving of those instances can be found in IsingFn.m.

Similarly, the simulator adaptation for a given instance iterates over $W$ and $V$ and converts their integer values to gates, according to equation 52. However, it also appends $H^{\otimes n}$ at the beginning and end in order to use equation 32 and also expands $\Lambda S$ using the gadget from equation 47. Then it makes use of the simulator functions: GadgetizeCircuit to convert the T-gates into the post-selection gadgets defined in figure 3, iteratively calls ChooseSubspace until the fidelity has reached the required value, and finally ComputeStabilizerGroup and NormEstimate to get the numerator and denominator from equation 49. Full code can be found in IsingFastFn.m.

To complete the first experiment, we combine the generation, exact, and approximate solutions in IsingBruteforceVsFast.m. There we generate 50 random instances with different numbers of
7. EXPERIMENTS

Figure 4: Comparison between true bruteforce outputs and averaged simulator outputs for 50 random Ising model instances with different numbers of qubits and T-gate counts.

qubits $n$ and T-gate counts $t$, solve them with the bruteforce, and 30 times each with the simulator. Then for each instance we take the mean and standard deviation of the 30 approximations as the single answer produced by the simulator and compare it to the true exact answer. Results are presented in figure 4. We see in subfigure (a) that occasionally there are clear outliers (plotted in red) when the simulator did not perform well. Because of its probabilistic and random nature - in picking the postselection strings $y$, generating the subspace $L$, generating the random stabilizer states $|\phi\rangle$, etc. - the simulator has some non-zero probability of returning a very different result from the true value. When those outliers are discarded, figure 4b shows that the approximated results match the true bruteforce values with very high precision - $R^2$ value of 0.9572. We conclude that the simulator has a $\sim 15\%$ chance to produce an outlier but is statistically significantly close to the true value in the other cases.

7.1.2 Perfect vs. noisy

Having verified the simulator’s correctness, we move onto the second part of the experiment. First we have to modify the random instances generator to restrict them to the NQIT architecture. We first fix the number of logical qubits to be used. Out of the 20 physical ones per ion trap, we dedicate 10 to entanglement distillation. As discussed in section 5.2, this way we have an estimation of the noise of two-qubits gates between different cells to be about the same as for two-qubit operations within a trap. Still, the overall noise cost of such inter-cell operations is higher because of the long time needed for distillation. Except for those 10 qubits per ion trap, we can consider all other physical qubits as logical ones. Therefore we are left with 20 traps of 10 logical qubits each, for a total of 200 application qubits. Note that this is significantly higher than the bruteforce upper limit size of $\sim 10$.

For this particular problem, fitting it to the networked structure means that we restrict the graph edges $w_{ij}$ so that there is exactly one connection between ion traps. That way there are no disjoint regions, which would effectively mean we are solving a number of different instances in parallel,
7. EXPERIMENTS

(a) Unrestricted 200 qubits.

(b) Restricted to a 4x5 grid, 10 qubits per cell.

Figure 5: Comparison between randomly generated unrestricted and restricted Ising model instances for 200 qubits. The graphs shown consider $W$ as an adjacency matrix and edge thickness corresponds to edge weights $W$. For illustration purposes, the unrestricted instance was generated with thresholds of 0.01 (no other restrictions), because otherwise too many edges overlap and visualization is poor. The main point is that the restricted case clearly shows the 2D grid network architecture of NQIT. It also has no unused disjoint qubits, unlike the unconstrained case.

instead of one big calculation. For the same reason we force the 10-edge sub-graphs within each trap to be connected. The vertex weights $v_k$ do not matter for the architecture fitting because they are single-qubit rotations, so we let them be fully random. Figure 5 shows an example comparison between an unrestricted and a restricted problem instance.

Finally, we want to bias the restricted instances towards a certain number of T-gates on average. This is needed because, given truly random $W$ and $V$, drawn from their respective integer domains, the expectation value for the number of T-gates would be $\langle t \rangle \approx \frac{200}{2} + 3 \cdot 20 \cdot \left(\frac{10}{2}\right) \cdot \frac{1}{2} \approx 1500$, using equation 48. The advanced simulator we are using, however, extends our classical capabilities to a few hundred qubits and Cliffords but only several dozen T-gates, since they are still exponentially hard. The original paper [38] reaches almost $t = 50$ in a single calculation. However, we need to run the simulation in a Monte-Carlo fashion - many times with different noises, in order to obtain estimates of its contribution. Such high T-gate counts then become unfeasible. We have decided to aim for $t \sim 15$ because it executes quickly enough to allow repetition while still being highly non-classical.

We achieve this T-count constraint in the following way. We introduce the $W_{\text{WeightThreshold}}$ and $W_{\text{WeightInterTrapThreshold}}$ parameters in the range $(0, 1]$. Then we randomly and independently assign $W$’s values according to the architectural constraints defined above. For each weight, a random number is generated in $(0, 1]$ and is compared to the relevant threshold parameter. If less than it, a random non-zero value in $w_{ij}$’s domain $\{1, 2, 3\}$ is assigned, otherwise it is assigned zero (i.e. no edge). Afterwards we iterate over the graph and add $w_{ij} = 2$ values wherever needed by the architecture (e.g. to make sure that exactly one edge exists between adjacent ion traps on the grid). By equation 48, those even values correspond to Cliffords and thus do not increase the T-gate count further. Finally, we iterate over all vertices and similarly assign random $v_k$ values whenever a randomly generated number is less than another parameter we have defined - $V_{\text{WeightThreshold}}$. 

29
7. EXPERIMENTS

Figure 6: Comparison between perfect and noisy aggregated results for random Ising model instances of 200 qubits in groups of 10 over a 4x5 ion trap grid. Every consecutive pair of a perfect mean (blue diamond) and a noisy mean (red square) describes one independent run. In order to show different runs, whose means are of different magnitudes, on the same plot, the means and standard deviations have been scaled by the perfect mean value of each run. Thus all the perfect means are equal to 1 in the plot. We observe that the noise generally increases the standard deviation and lowers the mean answer. The latter is explained by the fact that noise tends to equalize all output amplitudes to their equal prior probability values $2^{-n}$ for n qubits.

For the particular problem size we are considering - 4x5 grid of ion traps with 10 logical qubits in each - threshold values of 0.01 for all 3 parameters yielded average T-gate counts of between 10 and 20, as desired. The code for generating the restricted instances can be found in `IsingClusters.m`.

Finally, we need to provide a random source of noise to the restricted problem instances. This is an example of practically applying the more abstract description in section 6.3. We first add preparation and measurement errors (Pauli Xs) respectively to the front and back of the full circuit unitary. Each of them has a probability given by its rate (listed in section 5.2) to add a noise gate to any of the qubits. Then we construct the actual unitary in the middle by going through the $W$ matrix. For each non-zero edge $w_{ij}$ we randomly add two-qubit noise gates to the corresponding $AS^{w_{ij}}$ operation. Then we consider the time for its execution and independently add the random time-based noise (i.e. depolarization and dephasing onto all qubits). Finally, we analogously iterate over the $V$ vector, randomly add its single-qubit gate and time-based noises. The code that randomly adds noise to a given restricted instance can be found in `IsingFastNoise.m`.

We have so far presented all preliminary procedures needed to achieve our final goal for this particular problem - run experiments that compare perfect to noisy results for its random instances. Each single run is structured as follows:

- Generate a random restricted instance of the problem.
- Solve it 20 times using the perfect simulator.
- Solve it 20 times using the noisy simulator, i.e. adding random noise each time.
- Compare the means and standard deviations of the two result vectors of size 20.

We have presented results for 15 such independent runs in figure 6. Each run has 200 qubits in groups of 10 over a 4x5 ion trap grid. On average, each run had 1200 gates, of which $\sim 15$ are T-gates, and $\sim 150$ are added random noises. Every individual run corresponds to one random Ising model instance which has its own true partition function value that we are trying to approximate.
Those true values can vary in orders of magnitudes between different instances. Therefore, in order to show different runs on the same plot, we have first scaled for each run the four outputs - perfect mean and standard deviation over the 20 perfect solutions, and similarly noisy mean and standard deviation over the 20 noisy solutions - by dividing them all by the perfect mean. This allows us to illustrate visually the result from this experiment. We observe that noise increases the standard deviation of the outputs, i.e. they are noisier, as expected. Another remark is that the noisy means, compared to the perfect ones, are more often smaller rather than bigger. We explain this with the fact that noise tends to destroy information and so the more error in the system, the closer all amplitudes are to their equal prior probability values $2^{-n}$ for n qubits. Putting those qualitative observations into particular numbers, the mean values differ by 10% on average, and the standard deviations double when noise is added.

7.1.3 Summary

In this experiment we considered the problem of estimating the partition functions of random instances of the 2D complex-temperature Ising model. First we established the accuracy of the simulator by comparing its average outputs to exact bruteforce answers. This showed a $\sim 15\%$ chance of producing an insignificant outlier but otherwise strong correlation between the two with $R^2 = 0.9572$. After that we restricted the Ising model instances to the NQIT architecture and added its noise to the simulation. Comparing perfect to noisy simulations showed that the noise shifts the mean outputs by $\sim 10\%$ and approximately doubles their standard deviations.

7.2 X-programs in MBQC

The second example we consider is solving X-programs, defined in section 3.1, in the MBQC model from section 2.10. Similarly to the hypothesis test problem in section 3.2, the action $\theta = \pi/8$ is set to constant for all terms, which preserves the computational hardness of the problem. Being in IQP, X-programs do not have temporal structure. Therefore their corresponding MBQC patterns will have no depth, i.e. the full calculation can be done in two steps - entangling the resource state, and measuring all operation qubits. No two measurements depend on each other and therefore the execution is fully parallelizable. This is a desirable feature when considering noisy simulations because the shorter the execution time, the less corrupted outputs become through time-based noise channels such as depolarization and dephasing.

7.2.1 Bruteforce vs. simulator

Similarly to the previous example, we first have to establish how close our simulation outputs are to the true values, before introducing any noise. For constant $\theta = \pi/8$, each instance of the problem is fully defined by a binary matrix $P$ of size $O(n) \times n$. Columns represent the $n$ application qubits. Each row corresponds to one exponential term acting with Pauli Xs on qubits with value 1 in the row. For example,

$$P = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \quad \iff \quad C_X = \exp \left( i \frac{\pi}{8} X_1 X_3 \right) \exp \left( i \frac{\pi}{8} X_2 X_3 \right)$$

(53)
As before, the final answer we seek to estimate is $\langle 0^{\otimes n} | C_X | 0^{\otimes n} \rangle$. The bruteforce solution straightforwardly multiplies matrices according to the given matrix $P$, following equation 32. Its implementation can be found in `XprogramBruteforceFn.m`. Generating random unrestricted instances of the problem is trivial because it is equivalent to randomly populating an $O(n) \times n$ matrix with zeroes and ones. However, we need to be able to control the expected number of T-gates, analogously to the previous example.

In order to find a way to control the average T-gates count, we first need to establish where this complexity comes from. Given the example X-program from equation 53, the corresponding MBQC pattern is given in figure 7a. Qubits $q_i$ on the left are the usual application qubits, while $g_j$ are additionally required gate qubits for the resource state. Every edge represents an entanglement link that needs to be created via a $\Lambda Z$. Because of IQP’s defining property that all operations commute, the pattern has no depth, i.e. all measurements can happen simultaneously. As a consequence, the graph in figure 7a is bipartite. Given the MBQC pattern, we can construct the corresponding circuit from figure 7b, as also explained in [44, 36, 45].

Note that we are still working in MBQC and not the circuit model of computation, despite the linguistically confusing fact that we are building a circuit. That is because we follow the MBQC methodology which only allows $\Lambda Z$s for entanglement and measurements in different bases. We first create a resource state with $\Lambda Z$s according to the pattern. Then we have to measure the gate qubits in the $\{|0_{\pi/8}\rangle, |1_{\pi/8}\rangle\}$ basis [45] which is defined as:

$$
\{|0_{\pi/8}\rangle, |1_{\pi/8}\rangle\} = \left\{ \frac{1}{\sqrt{2}} \left( e^{-i\pi/8} |+\rangle \pm e^{i\pi/8} |-\rangle \right) \right\}
$$

(54)
As described in section 2.8, measurements in bases other than the computational \{\ket{0}, \ket{1}\} are performed by first rotating to the new basis. The correct rotation for the \{\ket{0_\theta}, \ket{1_\theta}\} basis is given by \(HR_{-\theta}XR_\theta H\). However, we notice that

\[
R_{-\theta}XR_\theta = e^{-i\theta}XR_\theta
\]

where the global phase \(e^{-i\theta}\) can be dropped as usual. Therefore the correct rotation for our case \(\theta = \pi/8\) is

\[
HXR_{2\pi/8}H = HXR_{\pi/4}H = HXTH
\]

Note that in the circuit in figure 7 operations are applied from left to right, but in the opposite direction when listed as a matrix product in equation 56. Finally, we need to apply Hadamards to the application qubits in order to bring them back to computational basis. As explained in section 2.10 though, certain Pauli corrections might be needed after the measurements. When physically executing the computation on a quantum computer, those can be considered as classical post-processing of the outputs that simply relabels the states, and therefore the algorithm description is complete. As shown in [46] (fig. 9) though, we can incorporate the corrections straight into the circuit using quantum control. We achieve this by adding \(AXs\) according to the same pattern used to produce the resource state initially. Note that since those corrections do not need to be physically executed, because of their equivalence to classical post-processing, we consider them as perfect, i.e. do not add any noise to them. The code that employs the logic described above to solve unrestricted instances using the simulator is in \texttt{XprogramMBQCFn.m}.

The above description of how to convert a given X-program \(P\) to a particular circuit also gives us the key to controlling the T-gates count. We have shown that every individual exponential (row in \(P\)) corresponds exactly to \(t = 1\), and the number of application qubits has no effect on \(t\). Similarly to the previous example, we want T-gate counts of no more than 20 in order to achieve feasible runtimes while preserving non-classicality.

With this, we are ready to run the bruteforce vs. perfect simulator experiment for the X-programs in MBQC problem. The code that combines the generation, exact, and approximate solutions can be found in \texttt{XprogramBruteforceVsMBQC.m}. There we generate 50 random instances, get their true answer using bruteforce, and solve them with the simulator 20 times each. Each instance is created by randomly populating with binary values a matrix \(P\) of randomly picked dimensions in \([5, 15] \times [5, 12]\). This corresponds to \(n \in [5, 12]\) and \(t \in [5, 15]\) where bruteforce’s complexity comes from high \(ns\), and the simulator’s complexity comes from high \(ts\). Results are shown in figure 8a. Unlike in the previous example, there don’t seem to be clear outliers. Averaged simulator outputs exhibit strong correlation with the true values with \(R^2 = 0.9619\).

### 7.2.2 Perfect vs. noisy

After verifying that the perfect simulator correctly solves the X-programs, we can now consider it in the context of NQIT. Analogously to the previous example, we dedicate 10 qubits to entanglement distillation and are thus left with 10 qubits in each of the 20 traps. The main advantage of solving the X-programs in MBQC is the full parallelization allowed. Therefore, we try to restrict their instances to the NQIT architecture in such a way that the execution is still as quick as possible. Algorithm 2 shows the steps needed to entangle part of the resource state including one gate qubit and application
Figure 8: **Left:** comparison between true bruteforce outputs and averaged simulator outputs for 50 random X-programs with different numbers of qubits in [5,12] and T-gate counts in [5,15]. Strong correlation with the true values is observed with $R^2 = 0.9619$. **Right:** an example restricted MBQC pattern for 3 traps, where application qubits are on the left and gate qubits are on the right. Gate qubits are still physically in the cells with the other ones. Analogously continued for more cells by assigning one gate qubit for every two neighbouring cells.

qubits from two neighbouring traps. Using teleportation, we can achieve this by only distilling an entanglement link between the traps once. Assuming that a cell cannot simultaneously distill entanglement links with more than one other trap, generating resource states that include qubits from several traps will take a significantly long time. For this reason we restrict each exponential term to act on qubits in at most two neighbouring traps.

**Algorithm 2 EntangleTwoTraps** ($p, g, c, l_1, l_2, Q_1, Q_2$)

This algorithm constructs part of the resource state for a given gate qubit $g$ in trap 1 according to its corresponding row $p$ of the X-program $P$. $Q_1$ is the set of all qubits in cell 1 with $g, l_1 \in Q_1$. Analogously, $c, l_2 \in Q_2$. $c$ is the qubit that will eventually be used for measurement after $g$'s value is teleported there.

1: for all $q \in Q_1 : p(q) = 1$ do 
2: \hspace{1em} $\Lambda Z (g, q)$ 
3: end for 
4: $\Lambda Z (g, l_1)$ 
5: Distill a Bell pair between $l_1$ and $l_2$ 
6: Bell measurement on $(g, l_1)$ which teleports $g$ to $l_2$ 
7: SWAP $(c, l_2)$ 
8: for all $q \in Q_2 : p(q) = 1$ do 
9: \hspace{1em} $\Lambda Z (c, q)$ 
10: end for 

In this most optimal case when links between adjacent traps are generated only once for each pair, we cannot teleport a gate qubit back to its original cell $i$, and it stays in the $i+1$ trap. In terms of algorithm 2, this means that $g$ from trap one gets sent to $c$ in trap two and is not returned back to one. This does not constitute a problem because $g$ had already been entangled with qubits from
7. EXPERIMENTS

its own trap before being teleported. However, then the same procedure is applied to the following two traps - two and three - and c is already occupied in the second one, which effectively lowers its free qubits count $Q_2$ by one. As a consequence, each trap (ignoring the boundaries) needs to reserve two qubits - one it will teleport further, and one it will receive from its predecessor. This leaves 8 qubits free to be used as application ones. Moreover, considering the same physical restriction that a trap cannot distill entanglement with more than one of its neighbours simultaneously, a single trap can either receive or forward a gate qubit but cannot do both at the same time. Therefore the full graph generation can happen in two link distillation time steps in the following way. First, all traps at odd positions pass forward their gate qubits using algorithm 2. Then the same is repeated for all even cells (and thus odd ones receive). This two-step resource state generation is schematically shown in equation 57.

\[
1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow \ldots \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow \ldots
\]  

(57)

Under the imposed 1D nearest-neighbor condition for the traps, there is a natural way to map the X-programs to NQIT’s networked architecture. We set $P$’s row count to 20, which corresponds to $t = 20$ - one gate qubit for each of the 20 cells. Then we let each gate qubit act on a random subset of the application qubits in its own cell and (except for the last one) the following cell. By considering only one nearest neighbour per trap we have effectively stretched the 2D grid of cells to 1D. However, the calculation is still classically hard because the random edges between qubits in adjacent traps allow the resource state to have as many as $2 \times 8 = 16$ neighbours on the scale of qubits, and not traps. An example MBQC pattern for such restricted instances is given in figure 8b. The code that generates the random restricted instances can be found in XprogramClusters.m. In a similar fashion to the previous example, we have introduced a threshold variable $x \in (0, 1)$ that randomly biases the exponential terms to include smaller or bigger numbers of qubits on average.

Finally, we need to add noise to the simulation. Analogously to the previous example, we first randomly assign preparation and measurement errors at the beginning and end of the circuit. As explained above, we leave the $\Lambda X$ corrections from figure 7b perfect. Then we add both operation and time-based noise to the $\Lambda Z$s that construct the resource state. Utilizing the logic from equation 57, we add an extra time step needed for the entanglement distillation between the two steps. Finally, we add noise to the basis rotations of the gate qubits. The noisy simulation code is in XprogramMBQNoiseFn.m.

We ran experiments on random restricted models by combining the perfect and noisy solutions in XprogramPerfectVsNoisy.m. Initially we considered the largest system sizes that our mapping to NQIT allows - 20 ion traps with 8 application qubits in each. We ran 30 random tests by solving each 20 times perfectly and 20 times with random noise. Results were quite different from the ones in the similar comparison of the previous example though. Every single noisy output had a value of 0, while perfect outputs were distributed in the $(0, 1)$ range. By artificially varying the noise rate values from section 5.2, we observed that as soon as a single noise gate is added, the output goes to 0. Another investigation we performed was whether only a certain type of noise causes such a drastic effect. By allowing only one type of noise gate at a time (e.g. only Pauli-Z dephasing), we
observed the same result - any noise corrupted the output entirely.

We then ran similar experiments for smaller systems in order to establish how this noise susceptibility varies with size. Results in figure 9 show that the added error does not always set all outputs to 0 for small enough systems. The $R^2$ values indicate how the noisy outputs match the perfect ones less as the number of qubits is increased. Even in the smallest example - four cells with only two application qubits in each - the correlation between the two is not high. At 12 8-qubit cells, a bit more than half of all available qubits, the $R^2$ value has become negative, i.e. the noisy outputs correlate with the perfect ones worse than a constant would. Increasing the size further simply sets all values to zero, as already observed for the maximum size.

**7.2.3 Summary**

In this example we applied our simulation methods to random X-programs in MBQC. We described a way to convert an instance of the problem to an MBQC circuit that can be used in our simulator. Comparing true bruteforce outputs and perfect simulated ones, we verified the correctness of our method with strong correlation $R^2 = 0.9619$ between the two. Then we devised a way to restrict the instances to the NQIT architecture and added its noise. Comparing averaged perfect and noisy outputs indicated that noise strongly influences the computation. Even systems of very few qubits exhibited weak correlation between the two aggregates. Increasing the size decreased this correlation further until all noisy outputs became zeros at approximately half the maximum available capacity.
7. EXPERIMENTS

Figure 9: Comparison between aggregated perfect and noisy outputs for random restricted X-programs for four different system sizes ($a \times b$ means $a$ traps with $b$ application qubits per trap). Noisy outputs have been averaged after discarding values of exactly 0. Linear regression trend lines have a set intercept at $(0,0)$. Equal scaling between the two axes has intentionally been preserved in order to clearly show the slopes. The $R^2$ values show that the noisy outputs approximate the perfect ones less successfully as the system size grows.
8 Discussion

In this project we have considered the problem of classically simulating quantum computation while taking into consideration the architecture and noise of a particular quantum device. We have developed a general framework of how one could approach such a problem in a systematic manner, given a specific task and a quantum machine of interest. The main points are: first, create a generator of random instances of the problem and establish the simulation’s correctness by comparing with true bruteforce outputs for small enough calculations. Then, devise a sensible way to restrict the instances to the device’s architecture which straightforwardly also leads to the application of noise. Finally, compare aggregated results for perfect and averaged noisy random restricted instances in order to establish the calculation’s sensitivity to noise. Because of the many degrees of freedom, the last step effectively constitutes a nested Monte-Carlo simulation - one level for instances generation, and another for different noise contributions (derived from the same probabilistic model).

We have exemplified this methodology with two problems. In the first one we estimated partition functions of random instances of the 2D complex-temperature Ising model. We observed that noisy outputs have about twice the standard deviation of perfect ones and means are shifted \( \sim 10\% \) towards the equal prior probability amplitudes. Therefore, we expect physical executions of this problem on the NQIT device to be possible up to its full capacity. This makes it a good candidate for achieving quantum supremacy, given the large number of qubits and the freedom to increase the T-gate count far beyond the simulator’s capability. The second example problem considered random IQP X-programs in MBQC. Results were significantly different as we observed that even the slightest noise affected outputs drastically. Values tended to decrease towards their (exponentially small) equal priors even for few-qubit systems which made answers inaccurate for all calculation sizes.

One possible future development of this project is to compare circuit-based and MBQC solutions for the identical instances of the same problem. Note that, since the Ising model in our first example is in IQP, it can also be represented as an X-program from our second example. This duality has been described in [36]. One would then need to establish the map between the two sets of instances in order to compare results. However, when we restrict both problems to NQIT’s architecture, the resulting constrained subsets may overlap only partially or not at all. This is an interesting theoretical question that could be investigated for various architecture restriction schemes. Directly comparing results from the two approaches for the same problem can provide better understanding for why our two examples behaved so differently.

Another future direction to be considered is to match simulation results to theoretical expectations. A recent paper by Bremner, Montanaro, and Shepherd evaluates theoretically the effects of constraints and noise on sparse IQP [47]. Despite the different noise model they have used, their results could be adapted for comparison to simulations.

Recent work by Bermejo-Vega, et al. [48] discusses in more depth the relation between random instances and computational hardness. In our project we have used weaker general statements regarding complexity. Using results from [48], our simulations can be improved in (at least) two ways. One is to make better justified decisions about the imposed architectural constraints and another is to more precisely pick truly hard instances when randomizing.
9 References


9. REFERENCES


[40] “NQIT.” http://nqit.ox.ac.uk/.


