Efficient Simulation of Quantum Circuits using Tensor Networks: Approximating Grover's Algorithm as a Matrix Product Operator

Theoretical Physics MSci Research Project, 2023-24

Conor O'Sullivan

University of Bristol
School of Physics

H.H.Wills Physics Laboratory, BS8 1TL, UK

NAME:	Conor O'Sullivan
DEGREE COURSE:	Theoretical Physics
PROJECT TITLE:	Efficient Simulation of Quantum Circuits using Tensor Networks: Approximating Grover's Algorithm as a Matrix Product Operator
YEAR OF SUBMISSION:	2024
SUPERVISOR:	Dr. Alan Reynolds
NUMBER OF WORDS:	9074

Dedication

To my grandfather Desmond

Declaration

All code written for this project was written by us (Conor and Jack). We played an equal role in writing the `Julia` implementation and `Python` implementation of Grover's algorithm as an MPO. We played an equal role in collecting data from these implementations, and their analyses.

Acknowledgements

I would like to express a special thank you to Alan Reynolds for his continued support and supervision throughout this project, for broaching the topic of simulating quantum circuits using tensor networks, and suggestions on what to implement in Python.

I would also like to express my gratitude to Stephen Clark for his guidance and direction regarding tensor networks and their Julia implementation, and the relation between bond dimension and entanglement.

I must also acknowledge my project partner, Jack Dennehy, for his collaboration, motivation and energy from start to finish.

I would also like to thank the Algorithms Research Group for fruitful discussions on optimising tensor contractions.

Finally, I owe much to my loved ones, whose support allows me to thrive.

Abstract

Simulations of many-body quantum systems suffer from the curse of dimensionality, where the computational resources scale exponentially with the size of the system, preventing their efficient simulation. From the theory of tensor networks, the concepts of matrix product states (operators) have been shown to effectively compress the state (operator) of multiple qubits (gates) for a given quantum circuit in a tensor network form, permitting efficient simulation through tensor contractions. We apply the concept of matrix product operators (MPOs) to the n-qubit Grover's algorithm quantum circuit representation by treating successive gate layers as order-$2n$ MPOs and contracting over all MPOs. The maximum bond dimension $\chi_{\max}$ for the Grover MPO is found to remain constant with value $\chi_{\max} = 2$ regardless of the number of qubits n for SVD cutoff values greater than 10^{-14} when implemented in Python using the zip-up algorithm. Implementation in Julia is also treated, and the entanglement generating power of Grover's algorithm is concluded to be less than previously believed.

Contents

Chapter 1

Introduction

The many-body problem in classical and quantum physical systems is regarded as a central problem in modern physics [1], and has been a driving force for developments in theoretical physics. An important subset of the quantum many-body problem is quantum computing, using the interactions between multiple quantum bodies (qubits) and quantum operators (gates) to manipulate the system in such a way as to interpret a computation as having been performed [2].

Multiple algorithms constructed in this quantum framework have been proved to solve certain problems faster than the corresponding known classical algorithms for that problem. The 'difficulty' of solving a certain problem is the subject of the field of computational complexity in computer science. For classical computers, an algorithm that solves a problem is said to be *efficient* if it's computational complexity was polynomial in the size of the problem. The discovery of quantum algorithms including the Deutsch-Jozsa algorithm [3] and Shor's algorithm [4] for prime factoring proved the existence of quantum algorithms that solved a problem efficiently despite there being no known classical efficient algorithm for solving the same problems [2]. These quantum algorithms were said to exhibit exponential speedup over their known classical counterparts. Later discoveries of other quantum algorithms, like Grover's algorithm [5] for unstructured database searching, also exhibited polynomial speedups over their known classical counterparts. Since their discovery, significant efforts have been made to find other quantum 'advantaged' algorithms that provide speedup for a certain class of problem [6]. These efforts have coincided with strides in engineering physical quantum computers on which to implement these quantum algorithms [7]. This remains an area of intensive research.

The current state of quantum computing technology is described as the noisy intermediate-scale quantum (NISQ) era, with computers capable of running algorithms for 50 to 100 qubits with 'noise' preventing complete control over those qubits [8]. Current gate technologies for quantum measurement offer poor fidelities, and not enough qubits can currently be used for fault-tolerant techniques to be implemented [2]. These issues are likely to be remedied over time. In the meantime, it is sensible to consider which quantum algorithms are worth implementing on future quantum hardware. Quantum algorithms with little to no quantum 'advantage' may be simulable on the most powerful classical computers. Thus, finding the boundaries between quantum advantaged algorithms and classical algorithms for a class of problems is important.

Simulating a general quantum n-body system on classical computers is an exponentially hard task due to the number of parameters necessary to fully characterise a quantum system $\mathcal{O}(2^n)$ growing exponentially with the system size n. This is known as the *curse of dimensionality.* Since computational resources of space on a classical computer are limited, large quantum systems quickly become intractable to store in a classical computer, let alone evolve in a sensible amount of time. Evolving a many-body quantum system is necessary for simulating a quantum algorithm that performs quantum computations. Therefore, to study quantum algorithms without their implementation on a quantum

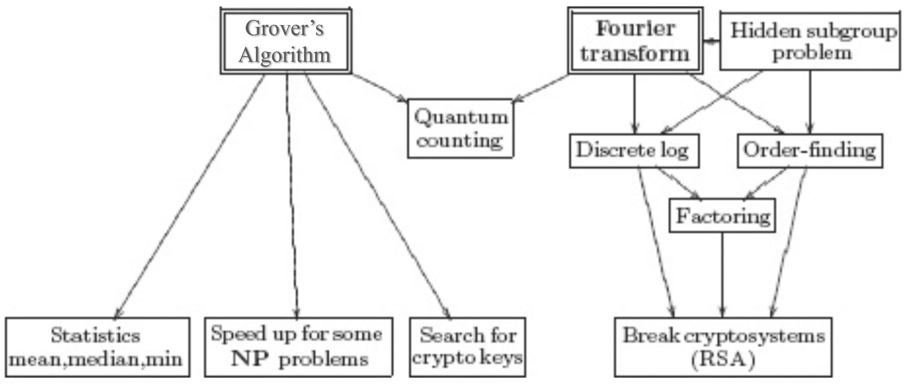


Figure 1.1: Diagram illustrating the problems whose quantum algorithms utilise the main subroutines of the quantum Fourier transform, and Grover's algorithm. Adapted from Ref. [2].

computer, one must settle for simulation of a small number of qubits on a classical computer.

However, tradeoffs exist between the accuracy of a simulation and the storage required for it. This is made possible by the exponential growth in parameters stemming from a matrix product structure required to combine quantum systems. Inherently, the useful information in this space is sparsely populated and is highly structured. Therefore, effective compressed representations that approximate the system are possible to construct, that scale in a more efficient way. Using these representations, larger qubit simulations become possible on classical hardware.

It has been shown that quantum algorithms can be studied using the quantum circuit representation [2]. These quantum algorithms may also be studied using tensor networks with an identical structure to the associated quantum circuit [9]. Since tensors can be exactly decomposed to matrix product representations [10, 11], and then further truncated to approximate matrix product representations, there exists a method to effectively compress a quantum algorithm from $\mathcal{O}(2^n)$ parameters to $\mathcal{O}(2n\chi_{\max}^2)$ parameters for some value $\chi_{\max}$. In this case, the approximate quantum algorithm may be simulated for higher qubit numbers on a given classical hardware, with up to exponential savings in space and time. However, this may not be the case for all quantum algorithms, especially those that feature high *entanglement*.

Quantum algorithms have quantum 'advantage' over their classical counterparts when they utilise the quantum properties of superposition and entanglement. The majority of quantum algorithms known for solving various problems rely on one of two main *subroutine* quantum algorithms: (1) the *quantum Fourier transform*, and (2) *Grover's algorithm* for unstructured database searching, as illustrated in Figure 1.1. Algorithms containing the quantum Fourier transform can experience exponential speedup over the corresponding classical algorithm, whilst those containing Grover's algorithm can experience a quadratic speedup.

Understanding which aspect of the structure of these quantum algorithms provides the quantum advantage is an important task for future algorithm development. The quantum Fourier transform was shown not to be the main cause of quantum advantage in quantum algorithms in which it was used as a subroutine [12], as it is efficiently simulable [13, 14] on a classical computer without approximation. When represented as a matrix product operator (MPO), the maximum bond dimension $\chi_{\max} = 8$ was required to incur no error beyond machine precision of 10^{-16} regardless of the number of qubits simulated [12].

In this project, Grover's algorithm was chosen to undergo investigation using the same methodology as in Ref. [12]. How well an MPO can approximate Grover's algorithm as a function of the number of qubits can inform us on the entanglement generating power of the algorithm, and ultimately if it is truly a quantum advantaged algorithm.

1.1 Outline

The relevant background for approaching this investigation is detailed in Chapter 2. This begins with an introduction to quantum information (Section 2.1) and quantum computation (Section 2.2). The use of quantum circuits (Section 2.3) to represent quantum algorithms (Section 2.4), notably Grover's algorithm, is illustrated. The concepts of tensors and tensor networks is then formalised (Section 2.5), as well as the concepts of exact and approximate matrix product representations of tensors. These are then related back to quantum algorithms, specifically Grover's algorithm (Section 2.6), as well as considerations of the mechanisms of entanglement that may hinder accurate representation.

Details of the method to implement simulations of the n-qubit Grover's algorithm are contained in Chapter 3, with the `Julia` implementation (Section 3.2) and `Python` implementation (Section 3.3) discussed separately.

Results and their discussion are provided in Chapter 4, with conclusions and future research directions stated in Chapter 5.

Chapter 2

Background

2.1 Quantum Information

The fundamental unit of classical information is the binary digit, or *bit*, which can take any value from the alphabet $\Sigma = \{0, 1\}$, therefore 0 or 1. Instructions and data can be encoded as a string of bits, forming the basis of classical computation, on which modern computers operate.

Creating an analogous model to classical computation in a system subject to quantum mechanics, the fundamental unit of information must be represented by some type of quantum state. A general quantum state

$$|\psi\rangle = \sum_{i=0}^{d-1} \alpha_i |i\rangle \tag{2.1}$$

that exists in a space spanned by the ortho-normal basis states $\{|0\rangle, |1\rangle, \ldots, |d-1\rangle\}$ can always be written as a superposition of all those basis states, weighted by some complex coefficients $\alpha_i \in \mathbb{C}$. The space in which the state $|\psi\rangle$ exists is known as the Hilbert space $\mathcal{H} = \mathbb{C}^d$, with dimension d equal to the number of ortho-normal basis states that span the space.

Since the size of the alphabet Σ for bits is two, it is sensible to consider a quantum state with Hilbert space dimension $d = 2$ (thus spanned by two ortho-normal states) for the fundamental unit of quantum information. This representation of a fundamental unit of quantum information is known as a quantum bit, or *qubit*.

2.1.1 Single Qubits

The state of a single qubit can be described as a vector $|x\rangle$ in the Hilbert space $\mathcal{H} = \mathbb{C}^2$ [2] where

$$|x\rangle = \sum_{i=0}^{1} \alpha_i |i\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle, \tag{2.2}$$

such that $\alpha_i \in \mathbb{C}$, $\sum_i |\alpha_i|^2 = 1$ and the two ortho-normal basis states are the computational basis states that can be written as

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \tag{2.3}$$

2.1.2 Multiple Qubit Systems

Analogously to how many bits together as a string were required for representing instructions and data in classical computation, many qubits must be considered together to represent useful quantum information.

In general, n quantum states $\{|\psi_j\rangle\}$ that exist in Hilbert spaces $\mathcal{H}_j = \mathbb{C}^{d_j}$ respectively can be considered as a single quantum state

$$|\Psi\rangle = \bigotimes_j |\psi_j\rangle = |\psi_1\rangle \otimes \ldots \otimes |\psi_n\rangle \tag{2.4}$$

that exists in the Hilbert space $\mathcal{H} = \bigotimes_j \mathcal{H}_j = \mathbb{C}^{d_1 \times \ldots \times d_n}$ created by the tensor product of all individual Hilbert spaces [2]. Likewise, the single quantum state for the whole system is described by the tensor product of the individual quantum states.

Applying this construction to qubits, a 1D many-body quantum system of n qubits can be described as a weighted superposition of the 2^n computational basis states $\{|00\ldots0\rangle, \ldots, |11\ldots1\rangle\}$, or a vector

$$|\Psi\rangle = \sum_{l=\{0,1\}^n} A_l |l\rangle \tag{2.5}$$

that exists in the Hilbert space $\mathcal{H}^{\otimes n} := \bigotimes_{j=1}^n \mathcal{H} = (\mathbb{C}^2)^{\otimes n} = \mathbb{C}^{2^n}$, spanned by the computational basis states, which is exponentially large in the number of qubits n [15]. Restricting this state to a single computational basis state,

$$|x\rangle = |x_1 x_2 \ldots x_n\rangle := |x_1\rangle \otimes \ldots \otimes |x_n\rangle, \tag{2.6}$$

this defines a quantum register of length n, where a classical bit string $x_1 x_2 \ldots x_n$ where $x_i \in \{0,1\}$ may be encoded in the total state of the system via $|x_i\rangle \equiv x_i$, and x_1 is the most significant bit.

For completeness, it should be noted that not all valid n-qubit states, that exist in $\mathcal{H}^{\otimes n}$, can be decomposed into a product state as in Eq. 2.6. States without a product state representation are said to be *entangled* states, whilst those that admit a product state representation are said to be *pure* states [2].

Tensor Products

The action of the tensor product operation $\otimes$ on two qubit states can be understood vectorally as the multiplication of the entire second vector (associated with the second qubit state) with each element in the first vector (associated with the first qubit state). For example,

$$|0\rangle \otimes |1\rangle = \begin{bmatrix}1\\0\end{bmatrix} \otimes \begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix} 1 \begin{bmatrix}0\\1\end{bmatrix} \\ 0 \begin{bmatrix}0\\1\end{bmatrix} \end{bmatrix} = \begin{bmatrix}0\\1\\0\\0\end{bmatrix}. \tag{2.7}$$

Notice the output vector of this operation has dimension $d = 4 = 2^2$, and thus is a vector in Hilbert space $\mathcal{H} = \mathbb{C}^4 = \mathbb{C}^{2\times2}$. Hence, an n-qubit system requires exponentially many parameters in the number of qubits to be fully characterised. Storing all these parameters on a classical computer is a task that becomes prohibitively hard for large quantum systems. This scheme also generalises to two quantum operators, where vectors are substituted for matrices.

2.2 Quantum Computation

A general classical computation can be interpreted as a function f that acts on an input bit string of length n and returns an output bit string of length m

$$f : x_{\text{input}} \in \{0,1\}^n \to x_{\text{output}} \in \{0,1\}^m, \tag{2.8}$$

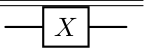
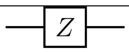
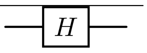
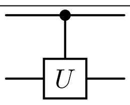
GATE	MATRIX	DIAGRAM
X	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$	
Z	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$	
H	$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$	
CU	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & u_{11} & u_{12} \\ 0 & 0 & u_{21} & u_{22} \end{bmatrix}$	

Table 2.1: Notable quantum gates, their matrix descriptions and their quantum circuit diagrams.

and f is then realised via a sequence of logic gates. Likewise, a quantum computation will involve transforming an input state $|x_{\text{input}}\rangle$ into an output state $|x_{\text{output}}\rangle$, both existing in $\mathcal{H}^{\otimes n}$, via some quantum operator $\hat{O}$ that acts on n qubits:

$$|x_{\text{output}}\rangle = \hat{O}|x_{\text{input}}\rangle. \tag{2.9}$$

This operator may be realised to an arbitrary accuracy by applying a sequence of *quantum gates* from a sufficiently large (universal) gate set $\mathcal{S}$ that act on the quantum information stored as qubits [2].

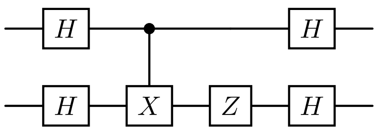
As a consequence of quantum mechanics, these gates must act linearly on a quantum state [2], and can be expressed as matrices with size exponential in the number of qubits they act on (2^m for an m-qubit gate). These matrices must also satisfy being unitary ($UU^\dagger = I$ for some matrix U) to preserve state normalisation after the transformation. Thus, any unitary matrix corresponds to a valid quantum gate, and vice versa.

Notable examples of quantum gates are listed in Table 2.1. The single qubit unitary gates include the X (or quantum NOT) gate that swaps the $|0\rangle$ and $|1\rangle$ states; the Z gate that inverts the coefficient of the $|1\rangle$ state; and the Hadamard H gate that creates a superposition. The Controlled-Unitary CU gate applies the unitary gate $U = \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{bmatrix}$ to the second qubit only when the first qubit is in the $|1\rangle$ state.

Quantum gates can be depicted as diagrams, as shown in Table 2.1, where the vertices correspond to the gate or its associated matrix, and the edges correspond to quantum wires transmitting qubit states.

2.3 Quantum Circuits

A sequence of quantum gates applied to their respective qubits for a multiple qubit system constitutes a quantum circuit. These circuits can be depicted in circuit diagrams. For example, the circuit 2.1 below



Circuit 2.1

depicts a two qubit system with Hadamard gates applied to each qubit, a controlled-X gate applied to the second qubit, a Z gate applied to the second qubit, followed by Hadamard gates applied to each qubit. Therefore, this circuit encodes the operator

$$\hat{O} = (H \otimes H)(I \otimes Z)(CX)(H \otimes H) \tag{2.10}$$

where the operator representation is written in the reverse order of the circuit diagram representation. When quantum circuits representing an operator $\hat{O}$ are applied to an initial qubit state, by convention the state $|0^{\otimes n}\rangle = |0\rangle \otimes \ldots \otimes |0\rangle$, an output state $\hat{O}|0^{\otimes n}\rangle$ is produced.

2.3.1 State Vector Representation

Using the matrix descriptions of quantum gates in Table 2.1 and the tensor product operation for matrices, Eq. 2.10 can be rewritten as a $(2^2 \times 2^2)$ matrix. In general, a quantum circuit acting on n qubits can be written as a $(2^n \times 2^n)$ matrix, whose size is thus exponential in the number of qubits it acts on. Hence, a complete description of an n-qubit system requires $\mathcal{O}(2^n)$ parameters or space, this description is known as the state vector representation.

Despite this requirement, the tensor product structure of these matrices entails they are sparse. This suggests that effective compressions of these matrices are possible.

2.4 Quantum Algorithms

The intention of quantum computers is to solve a problem (compute a desired output from some input) using less computational resources than possible when solving the same problem using a classical computer [2]. Speedups can be achieved over a classical computer when the quantum properties of superposition and entanglement are taken advantage of in the design of a quantum algorithm to solve a given problem.

A quantum algorithm that takes an initial multi-qubit state as an input and outputs the same qubits in a final state can be considered as a function `Alg()`, where

$$|x_{\text{output}}\rangle = \texttt{Alg}(|x_{\text{input}}\rangle) \tag{2.11}$$

and $|x_{\text{input}}\rangle, |x_{\text{output}}\rangle \in \mathcal{H}^{\otimes n}$. This is an identical description to a general quantum computation as in Eq. 2.9, and as such a quantum algorithm admits a quantum circuit representation for a sufficiently large gate set $\mathcal{S}$. Hence, quantum algorithms may be studied and simulated by constructing their associated quantum circuit.

2.4.1 Grover's Algorithm

Grover's algorithm is used for searching an unstructured database of N elements $(x_0, \ldots, x_{N-1})$ for a specific element, eg: x_0. It utilises an Oracle

$$f : \{x_0, \ldots, x_{N-1}\} \to \{0, 1\} \tag{2.12}$$

that can be queried with an element, and returns the value 1 only for the specific element, and 0 for all other elements [2]. The 'difficulty' of this problem is studied using *query complexity*, the number of queries needed to solve the problem. The best known classical algorithm for the same problem has a tight-bound query complexity linear in the number of elements to search $\Omega(N)$. This is improved on in Grover's algorithm to result in up to a quadratic speedup:

Theorem 2.4.1. *From Ref. [5], there exists a quantum algorithm that solves the unstructured search problem in $\mathcal{O}(\sqrt{N})$ queries with high probability.*

Let $N = 2^{(n+1)}$, where $(n+1)$ is the number of qubits required for grover's algorithm, thus the number of elements N to search through is exponential in the number of qubits. Assign an element to each computational basis, such that $x \in \{0,1\}^n \to |x\rangle$. Initialise the qubits in the state $|\psi\rangle = |0^{\otimes n}\rangle \otimes |1\rangle$. Let the Oracle operator O be defined by

$$O|x, z\rangle = |x\rangle|z \oplus f(x)\rangle, \tag{2.13}$$

and the Diffuser operator be defined by

$$D = |\psi\rangle\langle\psi| - I. \tag{2.14}$$

Repeated application of O and D to the superposition state

$$|\psi\rangle = \sum_{\{0,1\}^n} \frac{|x\rangle}{\sqrt{2^n}} \Big(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\Big), \tag{2.15}$$

created from applying Hadamard gates to all qubits in the initial state, allows the state to be rotated towards the $|x_0\rangle$ state encoding the desired element in the 2D plane spanned by the $|x_0\rangle$ and the state perpendicular to this (a superposition of all other basis states) $|x'\rangle$. This is illustrated in Figure 2.1. The number of times T that O and D need to be applied so that the probability $|\psi\rangle$ collapses to $|x_0\rangle$ upon measurement as intended approaches $\mathcal{O}(1)$ is $T = \frac{\pi}{4}\sqrt{N}$. Hence, $\mathcal{O}(\sqrt{N})$ oracle queries are required to recover the desired element x_0 with high probability.

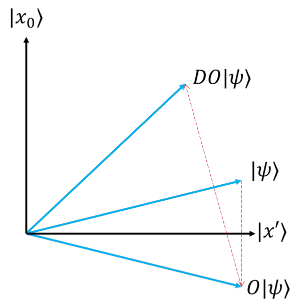
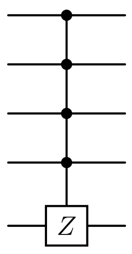


Figure 2.1: The action of one iteration of Grover's algorithm on a superposed state $|\psi\rangle$ spanned by computational basis states representing items to search.

Translating this to a quantum circuit representation of Grover's algorithm, the Oracle and Diffuser operators can be created using multi-qubit control gates [2]. While the oracle may be realised as a classical function implemented in a quantum circuit using uncomputation [2], in this project the oracle will be treated as a quantum gate.

Oracle Circuit

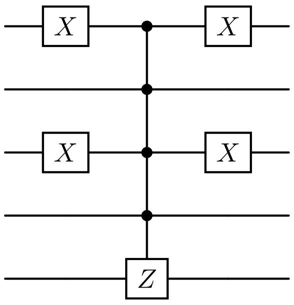
The operator O can be viewed as flipping the phase of the computational basis state associated with the desired search element. This logic can be encoded using multi-qubit controlled-Z gate:



This gate applies the Z gate to the last qubit if all control qubits and the target qubit are in the state $|1\rangle$,

$$|1111\rangle \rightarrow Z|1111\rangle = -|1111\rangle, \tag{2.16}$$

therefore flipping the phase of the 'desired' state $|1111\rangle$ that will be searched for. To change this 'desired' state to any other computational basis state, X gates can be applied to the qubits on either side of the multi-qubit controlled-Z gate. Eg: the circuit



flips the phase of the state $|0101\rangle$. Importantly, this conception of the Oracle requires the final qubit to be initialised in the $|1\rangle$ state, and to not contribute to encoding the search elements.

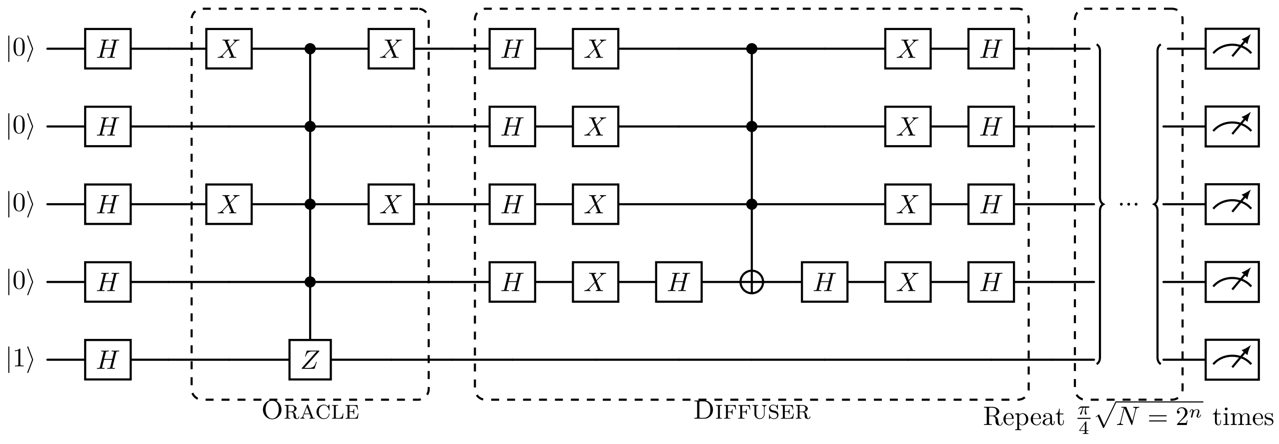


Figure 2.2: The 4-qubit Grover's algorithm circuit diagram. Note that although five qubits are used, the fifth qubit is used purely for realising the oracle, therefore this circuit can only search through all bit strings of length four. i.e. $N = 2^4 = 16$.

An example of Grover's algorithm as a quantum circuit with 4 qubits used for searching is shown in Figure 2.2. This is generalisable to n qubits for searching with the Oracle multi-qubit control gate always depending on all qubits, and the Diffuser multi-qubit control gate depending on all but the final qubit.

2.5 Tensor Networks

2.5.1 Tensors

Objects in linear algebra may be characterised by the number of indices necessary for their full description, as illustrated in Table 2.2. A scalar quantity contains one element, with no indices needed to distinguish between elements. A vector contains multiple elements in a row or column format, requiring one index to distinguish between elements, with the total number of elements determined

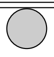
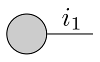
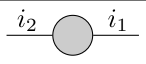
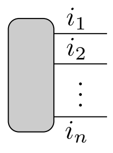
OBJECT	ORDER	INDEX SET $\mathcal{I}$	DIAGRAM
Scalar	0	$\emptyset$	
Vector	1	$\{i_1\}$	
Matrix	2	$\{i_1, i_2\}$	
General tensor	n	$\{i_1, \ldots, i_n\}$	

Table 2.2: Objects categorised by their order, and their diagram representations.

by the dimension of that index. Likewise, a matrix requires two indices to distinguish between elements, and the dimension of each index determines the shape of the matrix.

Generalising this, a tensor object may be described by its number of indices, known as its *order*, and the dimension of each index. Objects can be represented as diagrams, also shown in Table 2.2, as a vertex with the number of edges corresponding to the object's order.

Definition 2.5.1. An order-n tensor $\mathbf{T}$ is said to have an index set $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$, where $|\mathcal{I}| = n$, and $\dim(i_k) = d_k\ \forall k \in [1, n]$. In this case,

$$\mathbf{T} = [T_{i_1,i_2,\ldots,i_n}]_{i_1,i_2,\ldots,i_n} \tag{2.17}$$

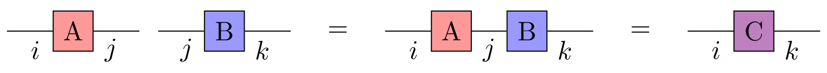
and exists in the space $\mathbb{C}^D$ where $D = \prod_k d_k$. Thus, the tensor $\mathbf{T}$ requires $\mathcal{O}(d^n)$ storage, where $d = \max_k d_k$, to completely specify all its parameters [15].

2.5.2 Tensor Contraction

The operations of matrix-vector and matrix-matrix multiplication can be considered as a contraction over a shared index between both objects. Eg: for two matrices $\mathbf{A}$ and $\mathbf{B}$,

$$\mathbf{AB} = \mathbf{C} \Leftrightarrow \sum_j A_{ij} B_{jk} = C_{ik}, \tag{2.18}$$

which may be represented diagrammatically as

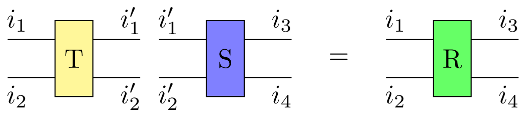


This operation may be generalised to contracting two tensor objects that share multiple indices, known as tensor contraction. For two tensors $\mathbf{P}$ and $\mathbf{Q}$, with index sets $\mathcal{I}$ and $\mathcal{J}$ respectively, that share the indices $\mathcal{I} \cap \mathcal{J}$ (where both the indices from each tensor that intend to be 'shared' have identical dimensions), their tensor contraction is given by

$$\sum_{\mathcal{I} \cap \mathcal{J}} P_{\mathcal{I}} Q_{\mathcal{J}}. \tag{2.19}$$

Eg: for two order-4 tensors $\mathbf{T}$ and $\mathbf{S}$,

$$\mathbf{TS} = \mathbf{R} \Leftrightarrow R_{i_1 i_2 i_3 i_4} = \sum_{i'_1 i'_2} T_{i_1 i_2 i'_1 i'_2} S_{i'_1 i'_2 i_3 i_4} \tag{2.20}$$



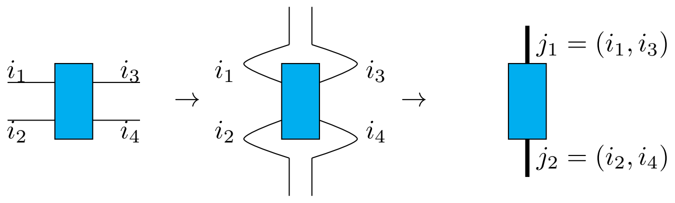
2.5.3 Tensor Networks

Having defined tensors and contractions between them, a tensor network can now be defined.

Definition 2.5.2. A tensor network is a set of tensors $\mathcal{T} = \{\mathbf{T}_1, \ldots, \mathbf{T}_N\}$ where each tensor shares at least one index with another.

2.5.4 Quantum Circuits as Tensor Networks

Observe that the number of required parameters to specify an n-qubit quantum gate, $\mathcal{O}(2^n)$, is identical to the number of parameters required to specify an order-n tensor where $\dim(i_k) = 2$ for all indices, $\mathcal{O}(2^n)$. Hence, a n-qubit quantum gate may be described by an order-$2n$ tensor. This may also be seen by the fact that size 2^n square matrices and order-n tensors of dimension 2 for all indices are the same object reshaped.



Quantum gates from Table 2.1 may now be represented as tensors, as in Table 2.3. Importantly, the connections between quantum gates in a quantum circuit diagram are preserved when representing gates as tensors. Here, quantum gates become tensors with the quantum wires between gates becoming shared indices between tensors.

Theorem 2.5.1. *From Ref. [9], a quantum algorithm with a quantum circuit C representation also admits a tensor network $\mathcal{T}$ representation with an identical structure.*

Therefore, quantum algorithms like Grover's algorithm can now be simulated and studied by constructing its tensor network from its quantum circuit representation.

Grover's Algorithm as a Tensor Network

Using the quantum circuit for Grover's algorithm, as seen for 4-qubits in Figure 2.2, the corresponding tensor network can be created with the same structure. This tensor network is illustrated in Figure 2.3.

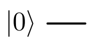
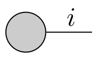
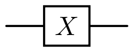
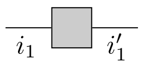
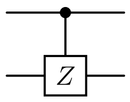
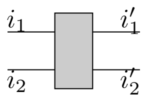
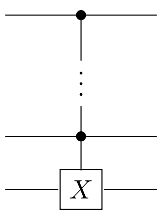
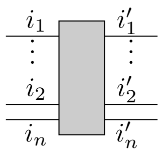
Quantum Object	Circuit Diagram	Tensor
Qubit state Eg: $\|0\rangle$		
Single-qubit gate Eg: X, Z, H		
Two-qubit gate Eg: CZ		
Multi-qubit gate Eg: C^mX		

Table 2.3: Quantum objects as represented in a circuit diagram and as tensors.

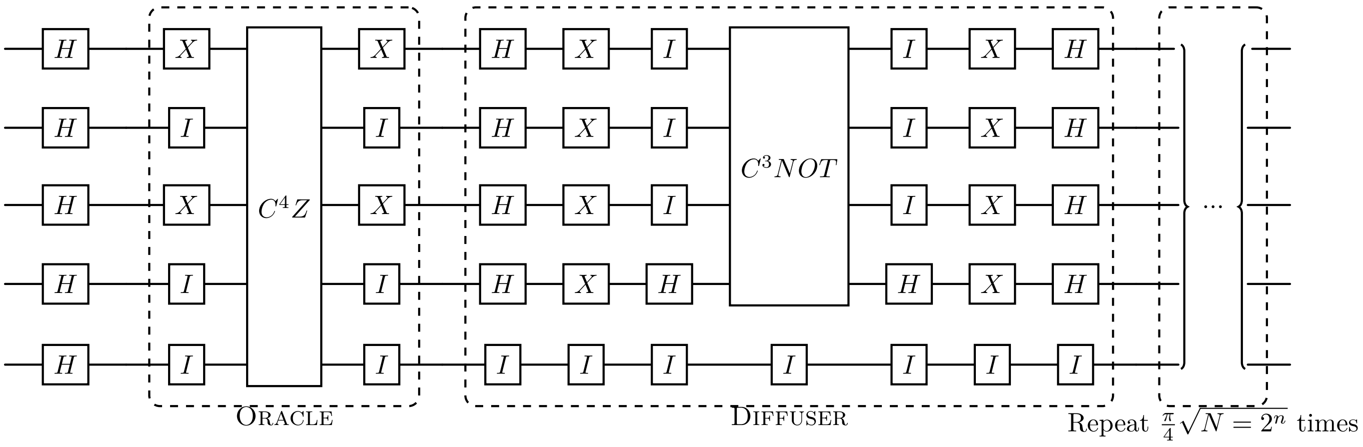


Figure 2.3: The 4-qubit Grover's algorithm's tensor network diagram. Here, the identity gate I has been made explicit.

2.5.5 Approximating Tensors using Matrix Product Representation

The *curse of dimensionality* entails a tensor with n indices cannot be handled by standard numerical methods, where memory use and amount of operations required to handle such tensors grows exponentially in n [10]. Hence, simulating and studying large quantum systems using tensor networks, instead of a state vector approach, remains prohibitively hard on a classical computer.

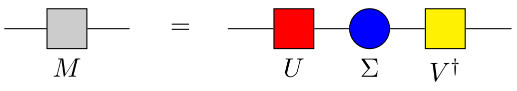
To alleviate this, efforts to find compressed but effective representations of such tensors have culminated in the *matrix product representation* [10, 16, 17, 18] of higher order tensors, after addressing issues alternative decompositions [19, 20, 21, 22] suffered from.

General tensors may have a *low-rank* structure that the MPS construction can take advantage of to create a compressed representation with limited error. Constructing an MPS from a tensor involves the procedure of singular value decomposition (SVD) on a tensor's associated reshaped matrix.

Singular Value Decomposition (SVD)

A real matrix $\mathbf{M}$ can always be decomposed into three matrices

$$\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \tag{2.21}$$



via singular value decomposition (SVD), where $\mathbf{U}$ and $\mathbf{V}$ are unitary (obey $\mathbf{U}\mathbf{U}^\dagger = \mathbf{V}\mathbf{V}^\dagger = \mathbf{I}$) whilst $\mathbf{\Sigma}$ is a diagonal matrix of real positive elements that may be ordered by descending size. The diagonal elements $\sigma_i = \mathbf{\Sigma}_{ii}$ are known as the singular values of $\mathbf{M}$, and the number of non-zero singular values $r = \text{rank}(\mathbf{M})$ equals the *rank* of $\mathbf{M}$.

An effective compressed representation of a matrix $\mathbf{M}$ can be achieved using a reduced SVD. The truncated SVD provides an optimal low-rank matrix approximation $\tilde{\mathbf{M}}$ of matrix $\mathbf{M}$ with fixed rank χ, where

$$\tilde{\mathbf{M}} = \mathbf{U}\tilde{\mathbf{\Sigma}}\mathbf{V}^{,} \tag{2.22}$$

and $\tilde{\mathbf{\Sigma}}$ is identical to $\mathbf{\Sigma}$ except it only contains the χ largest singular values. This conception is equivalent to minimising the Frobenius norm of the difference between $\tilde{\mathbf{M}}$ and $\mathbf{M}$

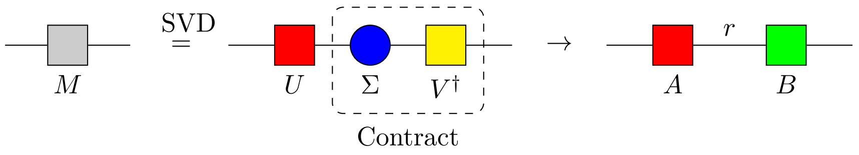
$$||\tilde{\mathbf{M}} - \mathbf{M}||_F^2 = \sum_{j=\chi}^{r} \sigma_j^2 = \sum_{j=\chi}^{r} \varepsilon_j^2, \text{ where } \chi \leq j \leq r, \tag{2.23}$$

which quantifies the error

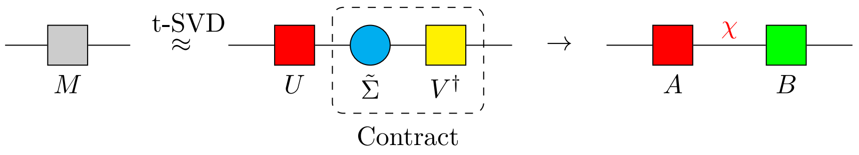
$$\varepsilon = \sum_j \varepsilon_j^2 \tag{2.24}$$

in the approximation of a matrix $\mathbf{M}$ as the sum of the squares of the singular values removed from its original $\mathbf{\Sigma}$ matrix [10].

Instead of three matrices, $\mathbf{M}$ can be decomposed to two matrices $\mathbf{A}$ and $\mathbf{B}$ by performing SVD and then contracting over the index shared by $\mathbf{\Sigma}$ and $\mathbf{V}^T$. This leads to two matrices $\mathbf{A} = \mathbf{U}$ and $\mathbf{B} = \mathbf{\Sigma}\mathbf{V}^T$ that share one index with dimension r.

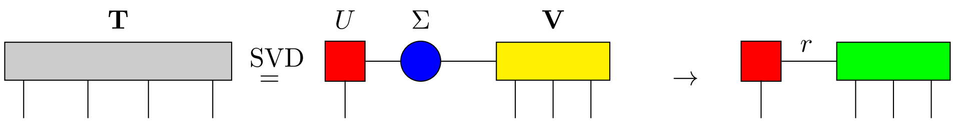


Were truncated-SVD to have been performed using $\tilde{\mathbf{\Sigma}}$, the matrix $\mathbf{B} = \tilde{\mathbf{\Sigma}}\mathbf{V}^T$ is changed and the shared index dimension is reduced to χ.

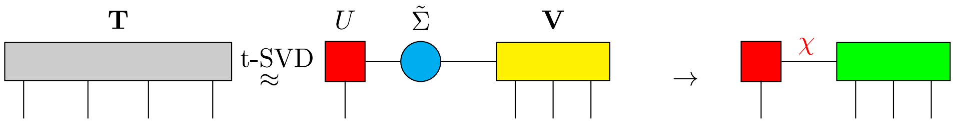


Generalising SVD to Tensors

As seen in Section 2.5.4, matrices can be considered as reshaped tensors. Hence, SVD can be considered to be performed on an order-n tensor $\mathbf{T}$ when generalising the matrix approach to tensors.



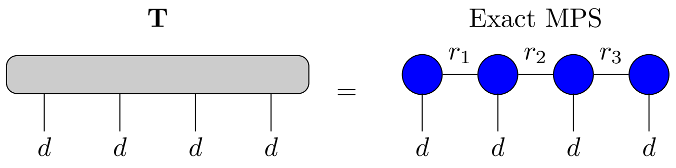
Likewise, truncated SVD can also be performed for tensor $\mathbf{T}$, leading to the approximate decomposition:



Matrix Product State (MPS) Representation

When an order-n tensor $\mathbf{T}$ is decomposed as above, SVD may be applied iteratively to the $\mathbf{V}$ tensor (and its resulting tensors) using the tensor train SVD (TTSVD) algorithm [10], until only order-2 and order-3 tensors remain.

Therefore, an order-n tensor $\mathbf{T}$, where all indices have dimension d, may be decomposed exactly [11, 10] into a tensor network of n smaller tensors, each with one 'real' index and at most two 'virtual' indices. These tensors are connected linearly through shared 'virtual' indices of dimension r_j that are sufficiently large as to exactly represent the original tensor. The dimension of these virtual indices is known as the *bond dimension*. This representation of an order-n tensor is known as a *matrix product state* (MPS).

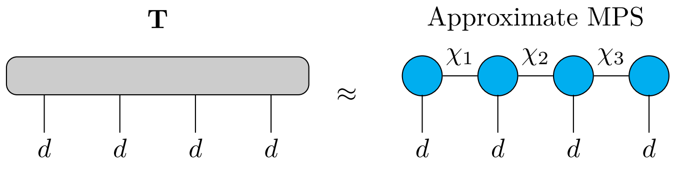


The MPS representation can be written as

$$T^{i_1,\ldots,i_n} = \sum_{r_1,\ldots,r_n} A^{i_1}_{r_1} A^{i_2}_{r_1,r_2} \ldots A^{i_{n-1}}_{r_{n-2},r_{n-1}} A^{i_n}_{r_{n-1}}, \tag{2.25}$$

where superscript indices refer to 'real' indices, and subscript indices refer to 'virtual' indices.

However, the approximate MPS, constructed from truncated SVD, with reduced bond dimensions $\{\chi_j\}$ will be more useful for the purposes of efficiently simulating Grover's algorithm.



Importantly, for sufficiently small maximum bond dimension $\chi_{\max} = \max_j \chi_j$, massive data compression can be achieved in characterising a tensor using the approximate MPS representation, where the number of parameters necessary changes as

$$\mathcal{O}(d^n) \to \mathcal{O}(nd\chi^2_{\max}). \tag{2.26}$$

Hence, storing these tensors changes from a task exponential in the number of qubits n to being linear in n and polynomial in $\chi_{\max}$, potentially providing exponential improvement for sufficiently small $\chi_{\max}$.

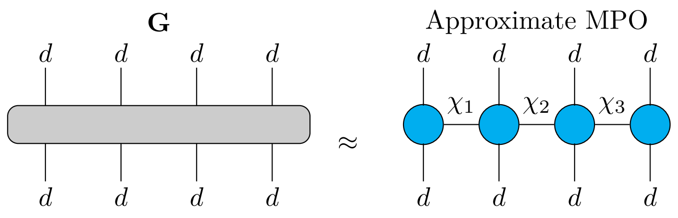
How accurately this MPS represents the original tensor $\mathbf{T}$ is a function of the maximum bond dimension $\chi_{\max}$, where the error introduced in the approximation is given by ε in Eq. 2.24. It may be the case that to maintain a given error of approximation of a specific tensor, the MPS representation requires a large bond dimension, and thus massive data compression cannot be achieved using MPS for that specific tensor. It will be shown that the maximum bond dimension $\chi_{\max}$ can be related to the entropy entanglement of the operator represented by the tensor $\mathbf{T}$.

Matrix Product Operator (MPO) Representation

It will also be useful to define an exact matrix product operator (MPO) representation of an order-$2n$ tensor $\mathbf{G}$, which can be written as

$$G^{i_1,\dots,i_n,i'_1,\dots,i'_n} = \sum_{r_1,\dots,r_n} B^{i_1,i'_1}_{r_1} B^{i_2,i'_2}_{r_1,r_2} \dots B^{i_{n-1},i'_{n-1}}_{r_{n-2},r_{n-1}} B^{i_n,i'_n}_{r_{n-1}}, \tag{2.27}$$

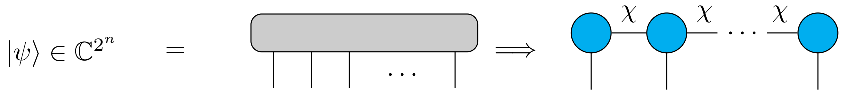
and can be constructed similarly to the MPS. Likewise, it will be more useful to consider the approximate MPO, with bond dimensions $\{\chi_j\}$, for simulation purposes.



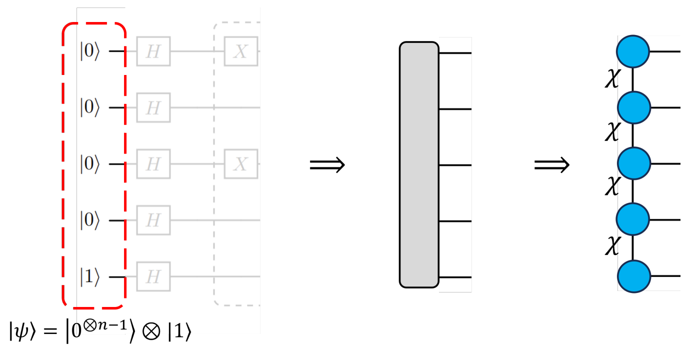
2.6 Quantum Algorithms as Matrix Product Representations

2.6.1 Quantum Objects as Matrix Product Representations

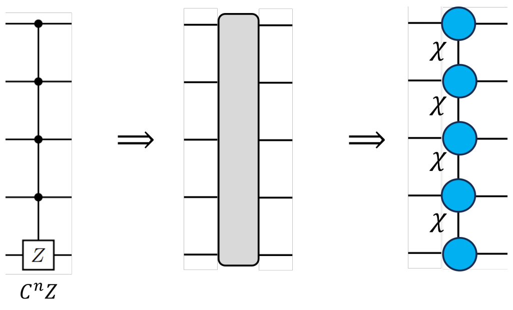
An n-qubit quantum state $|\psi\rangle$ can be represented via an order-n tensor. This tensor may be decomposed exactly to an MPS with sufficiently large maximum bond dimension χ, and all 'real' indices dimensions $d = 2$.



A relevant example for simulating Grover's algorithm is the input qubit state $|\psi\rangle = |0^{\otimes n-1}\rangle \otimes |1\rangle$. Likewise, the output qubit state after Grover's algorithm has been applied to the input qubit state can also be decomposed exactly as an MPS.



An n-qubit gate can be represented via an order-$2n$ tensor. This tensor may be decomposed exactly to an MPO with sufficiently large bond dimension χ, and all 'real' indices dimensions $d = 2$. A relevant example to simulating Grover's algorithm is the $C^n Z$ gate:

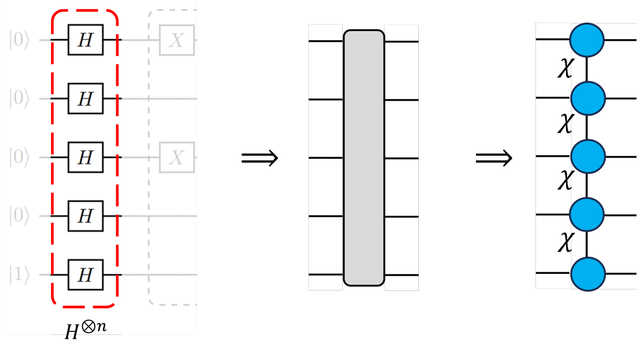


2.6.2 Quantum Circuits as Matrix Product Representations

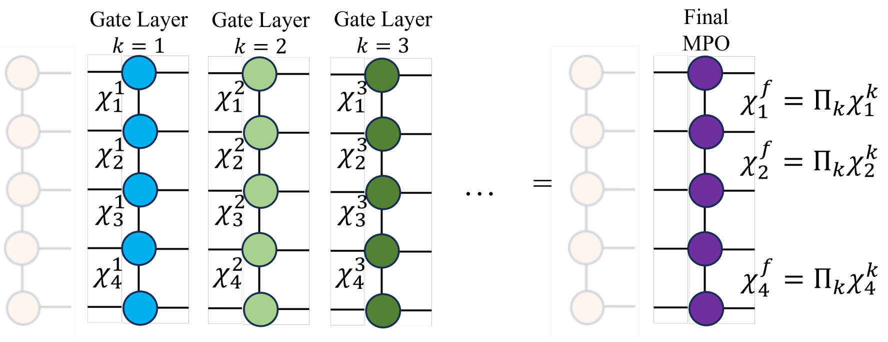
Recall from Section 2.5.4 that a quantum algorithm that acts on n qubits may be represented by a tensor network with n input indices and n output indices.

Contracting through all shared indices in the tensor network would yield an order-$2n$ tensor representing the circuit. This tensor could then be decomposed into an MPO representation.

Alternatively, consider each layer in this tensor network individually. By approximating each layer with an MPO, a new tensor network consisting purely of MPOs can be constructed that approximates the original.



Through contraction of adjacent MPO indices, a final MPO that represents the quantum algorithm may be constructed.



This final MPO will have bond dimensions $\{\chi_j\}$ where $\chi_j = \prod_k \chi_j^k$ and χ_j^k refers to the jth bond dimension of the MPO for the kth gate layer.

Since the outcome of tensor contractions does not depend on the contraction order [9], the MPO constructed via the former procedure should be identical to the MPO constructed via the latter for a given χ. The latter procedure was utilised in the `Julia` code, while the former procedure was utilised in the `Python` code.

2.6.3 Approximability by Matrix Product Representation

Recall from Section 2.5.5 that an MPS (MPO) can approximate an order-n (order-$2n$) tensor to within error ε for a given maximum bond dimension $\chi_{\max}$ to produce up to exponential improvements in storage.

Naturally, one may ask if all quantum systems written as tensors could benefit from this scheme? In other words, could quantum systems specified in $\mathcal{O}(2^n)$ parameters be compressed to $\mathcal{O}(2n\chi_{\max}^2)$ parameters using matrix product representation to a good approximation?

Were this to be true for all quantum systems, it would appear unnecessary to construct physical quantum computers to run quantum algorithms, when they could all be simulated on a classical computer to within high accuracy whilst scaling linearly with the number of qubits.

Unfortunately, this is not the case for all quantum systems and algorithms. This is due to the quantum property of *entanglement* that can be present in quantum systems, and possible to generate in quantum algorithms.

Entanglement

Entanglement is a property of bipartite quantum systems, consisting of subsystems A and B, whose state $|\Psi\rangle$ exists in the Hilbert space $\mathcal{H} = \mathcal{H}_A \otimes \mathcal{H}_B$, where $\dim \mathcal{H}_A = \dim \mathcal{H}_B = n$, constructed from the tensor product of Hilbert spaces of each subsystem. Only pure quantum states will be considered in this project, for which sharp distinctions can be drawn between entangled and unentangled states [23].

Definition 2.6.1. From [23], a pure state is entangled if and only if its state vector $|\Phi\rangle$ cannot be expressed as a tensor product of its subsystem states $|\Phi_A\rangle \otimes |\Phi_B\rangle$.

A pure state's entanglement is measured by its *entropy of entanglement*,

$$E(|\Psi\rangle) = S(\rho_A) = S(\rho_B), \tag{2.28}$$

which equals the apparent von Neumann entropy of either the A or B subsystem considered individually. The von Neumann entropy S of a state $|\Phi\rangle$ is written in terms of the state's density matrix $\rho = |\Phi\rangle\langle\Phi|$ as

$$S(\rho) = -\mathrm{Tr}(\rho \log_2 \rho). \tag{2.29}$$

Rather than the full density matrix, the entropy of entanglement takes as argument the reduced density matrix for subsystem A, $\rho_A = \mathrm{Tr}_B(|\Psi\rangle\langle\Psi|)$, which is found by tracing the whole system's density matrix ρ over subsystem B's degrees of freedom [23]. The same is true for ρ_B.

The entropy of entanglement can take values $E \in [0, \log n]$, with zero implying a product state structure, and $\log n$ implying a maximally-entangled state of two n-state particles.

Operators can also have their entropy of entanglement quantified by substituting ρ with the operator $\hat{O}$ in Eq. 2.29.

A pure state $|\Psi\rangle \in \mathcal{H}^{\otimes n} = \mathbb{C}^{2^n}$ of n qubits, partitioned into disjoint subsets of qubits A and B, can be rewritten as a Schmidt decomposition with respect to the partition A:B as

$$|\Psi(A,B)\rangle = \sum_{l=1}^{\chi_A} s_l |\Psi_A^l\rangle \otimes |\Psi_B^l\rangle, \tag{2.30}$$

where $\{|\Psi_A^l\rangle\}$ are ortho-normal Schmidt basis states and eigenvectors of the reduced density matrix ρ_A, and likewise for $\{|\Psi_B^l\rangle\}$ [11]. The coefficients $\{s_l\}$ are the Schmidt coefficients. The von Neumann entropy for $|\Psi\rangle$ is then

$$S(|\Psi\rangle\langle\Psi|) = -\sum_{l=1}^{\chi_A} s_l^2 \log s_l^2. \tag{2.31}$$

When decomposed this way, the rank χ_A of ρ_A is a natural measurement of entanglement between the qubits in A and qubits in B. Therefore, to quantify the entanglement of a state $|\Psi\rangle$, the maximal value of χ_A over all possible bipartite splittings A:B of the n qubits can be used:

$$\chi = \max_A \chi_A. \tag{2.32}$$

The entropy of entanglement is given by $E_\chi = \log_2 \chi$, since $E_\chi \in [0, n/2]$.

Entanglement and Matrix Product Representation

Importantly, an MPS can always be rewritten in Schmidt decomposition form [24]. Choosing an orthogonality centre at a given MPS site and observing an adjacent site, the Schmidt rank will be given by the bond dimension between those two sites. Furthermore, performing SVD about the orthogonality site will return a $\mathbf{\Sigma}$ with singular values σ_l equal to the Schmidt coefficients s_l [12, 24, 25].

Therefore, the MPS bond dimensions are related to the entanglement present between the two subsystems connected via that bond. As such, the maximum MPS bond dimension $\chi_{\max}$ quantifies the entanglement of the quantum state it approximates.

Similarly, an n-qubit quantum gate operator U acting on a bipartite Hilbert space $\mathcal{H} = \mathcal{H}_A \otimes \mathcal{H}_{\mathcal{B}}$, where $\dim \mathcal{H}_A = \dim \mathcal{H}_B = n$, can be written as an operator Schmidt decomposition as

$$U = \sum_{m=1}^{\chi_A} s_m A_m \otimes B_m \tag{2.33}$$

where $\{A_m\}$ and $\{B_m\}$ are operators acting on subspaces $\mathcal{H}_A$ and $\mathcal{H}_B$ respectively [12]. An MPO can always be written in operator Schmidt decomposition form, and the maximum bond dimension quantifies the entanglement generating power of the operator the MPO approximates.

Bond Dimension and Approximability

When approximating the operator that represents a quantum algorithm as an MPO, the maximum bond dimension $\chi_{\max}$ is constrained to the maximum number of Schmidt coefficients required due to the entanglement generating power of the operator.

Since the Schmidt coefficients are arranged in descending order in the diagonal matrix $\mathbf{\Sigma}$, the rate of decay of adjacent coefficients determines how many singular values there can be for a given cutoff value ε. This directly determines the bond dimension χ for each bond in the MPO, and therefore determines the size of χ_{max}. It happens that for Schmidt coefficients decaying at least as fast as an exponential, the operator can generate at most a constant amount of entanglement irrespective of the number of qubits used [12].

Observing how the maximum bond dimension χ_{max} scales with number of qubits n, or MPO sites, can indicate whether a quantum algorithm is simulable on a classical computer. For diverging χ_{max} with n, this suggests the entanglement generating power of the algorithm is large, and thus cannot be well approximated by an MPO to realise data compression. Observing the decay rate of the Schmidt coefficients can indicate the approximability of Grover's algorithm as an MPO as a function of the number of qubits.

Chapter 3

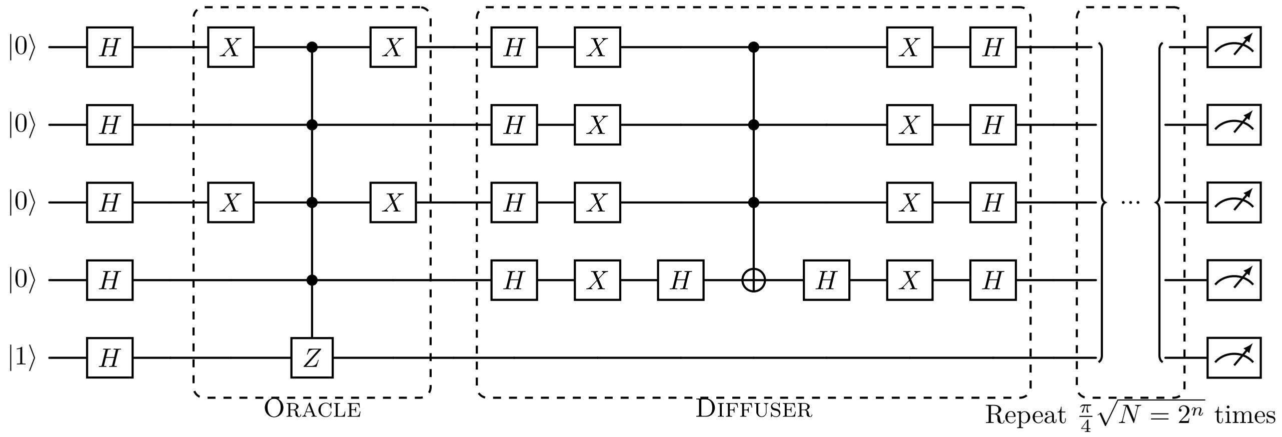
Method

The n qubit Grover's algorithm is to be simulated using an approximate n site MPO representation, constructed from contraction of approximate MPOs for each gate layer. The success of applying the Grover MPO on an initial qubit state, represented as an MPS, for a given oracle will be verified by contracting the MPO with the known final state MPS and assessing the scalar value, which should be the probability amplitude of the circuit outputting that final state.

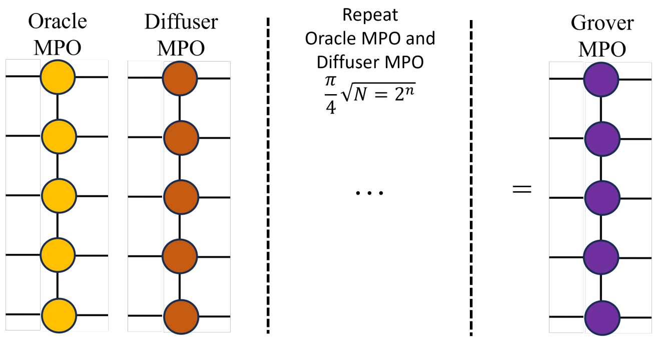
Given the MPO accurately represents Grover's algorithm, maximum bond dimension $\chi_{\max}$ of the Grover MPO will be assessed, and how maximum bond dimension $\chi_{\max}$ scales with the number of qubits n can be observed.

3.1 Grover's Algorithm as an MPO

Recall that Grover's algorithm represented as a quantum circuit appears as



To simulate this circuit for n qubits, the circuit can be separated to an MPS representing the initial qubit states, and an MPO representing the rest of the circuit. Since Grover's circuit consists of the repeating structures ORACLE and DIFFUSER, constructing an overall Grover MPO will require constructing an Oracle MPO and a Diffuser MPO, with repeated contraction between these.


Oracle MPO
Diffuser MPO
Repeat Oracle MPO and Diffuser MPO $\frac{\pi}{4}\sqrt{N=2^n}$
...
=
Grover MPO

This construction was implemented in two settings: (1) In the language `Julia` using the `ITensor` library [26], (2) In the language `Python` using the `NumPy` library.

3.2 Implementation: Julia and ITensor

The `ITensor` package for `Julia` was created to automate tensor network manipulations, and has in-built MPS / MPO objects for simulating quantum systems.

For an N qubit simulation, the matrix product representation skeleton can be created using the `siteinds()` function,

```
MPS_sites = siteinds("Qubit", N)
```

where the site type `"Qubit"` is specified as an argument so that the index dimensions will have dimensions $d = 2$.

The initial qubit state $|0^{\otimes N-1}\rangle \otimes |1\rangle$ can then be created as an MPS using the following logic and the `MPS()` function:

```
state = [n==N ? "1" : "0" for n=1:N]
psi_0 = MPS(MPS_sites, state)
```

The layer of Hadamard gates could be created as a single N qubit operator using the `OpSum()` function:

```
opsum_H1 = OpSum()
for j=1:N
    opsum_H1 += "H", j
end
```

Here, `OpSum()` acts as a list of gate types (in this case, `"H"` for Hadamard gates) applied to a specific matrix product site `j`. This operator can be converted to an MPO using the `MPO()` function:

```
MPO_H1 = MPO(opsum_H1, MPS_sites)
```

Oracle MPO

The Oracle MPO is constructed for easy repetition later in the circuit. It's construction involves the `OpSum()` and `MPO()` functions for the gate layers surrounding the multi-qubit controlled-Z gate.

```
opsum_X1 = OpSum()
# Add gates here...
for k=1:N
```

```
        opsum_X1 += "I", k
    end
    opsum_X1 += "X", 2
    #... end of gates added
    MPO_X1 = MPO(opsum_X1, MPS_sites)
```

This applies the identity `"I"` gate to all qubits, followed by X gates at all specified positions. This implemented the specific Oracle for selecting the $|101\ldots1\rangle$ state, however any Oracle can be used by changing the number and positions of `"X"` gates in the `OpSum()`. Again, the `OpSum()` is converted to an MPO.

The multi-qubit controlled-Z gate that acts on all qubits, $\mathrm{C}^N Z$, is then constructed by reshaping its matrix representation (see Appendix A) for matrix construction).

```
    M = Matrix(I, 2^N, 2^N)
    M = Int.(M)
    M[2^N, 2^N] = -1
    CZ = ITensor(M, MPS_sites, MPS_sites') #order-2N tensor

    XCZ = apply(CZ, MPO_X1, cutoff=C0)
```

The matrix `M` is reshaped to an order-$2N$ tensor using the `ITensor()` function, using the indices sets `MPS sites` and `MPS sites'` for input and output indices. This `ITensor` object is then applied to the previous gate layer MPO, and converted to an MPO via iterative truncated SVD decomposition, with the cutoff value for truncation specified by the argument `cutoff`. Applying the previous gate layer MPO again creates the Oracle MPO

```
    MPO_ORAC = apply(MPO_X1, XCZ, cutoff=C0)
```

Diffuser MPO

The diffuser MPO is also constructed for easy repetition later. Similar uses of the `OpSum()` and `MPO()` functions are used to construct MPOs for the gate layers either side of multi-qubit controlled-X gate. The multi-qubit controlled-X gate, $\mathrm{C}^{N-1}X$, was constructed in the same way as the $\mathrm{C}^N Z$ gate, by reshaping its matrix representation. The matrix representation was constructed as follows (see Appendix A) for matrix construction):

```
    M2 = Matrix(I, 2^N, 2^N)
    M2 = Int.(M2)

    if N==2
        M2[2^N, 2^N] = 0
        M2[2^N-1, 2^N-1] = 0

        M2[2^N, 2^N-1] = 1
        M2[2^N-1, 2^N] = 1
    else

        M2[2^N, 2^N] = 0
        M2[2^N-1, 2^N-1] = 0
        M2[2^N-2, 2^N-2] = 0
        M2[2^N-3, 2^N-3] = 0

        M2[2^N, 2^N-2] = 1
        M2[2^N-2, 2^N] = 1

    M2[2^N-1, 2^N-3] = 1
    M2[2^N-3, 2^N-1] = 1
end

CNOT = ITensor(M2, MPS_sites, MPS_sites')
```

In this rudimentary approach, matrix elements were set manually according to the known final matrix, which could then be reshaped to an order-$2N$ tensor using the `ITensor()` function.

Applying all these gate layer MPO's to each other created the Diffuser MPO.

```
MPO_DIFF = apply(MPO_H2, H3CNOTH3X2, cutoff=CO)
```

Repetition of Oracle and Diffuser MPOs

The number of repetitions `Rep` was computed. An identity MPO `MPO TOT` was then created using the `OpSum()` function to be updated by the repetitions of the Oracle and Diffuser MPOs.

```
for i=1:Rep
    MPO_TOT = apply(MPO_ORAC, MPO_TOT)
    MPO_TOT = apply(MPO_DIFF, MPO_TOT)
    @show MPO_TOT
end
```

The Grover MPO is equal to the final `MPO TOT` outputted from this for-loop. The dimensions of the 'real' and 'virtual' bond indices are printed in the console for this MPO, and were recorded.

Calculating the von Neumann Entropy from the Grover MPO

By defining a function `entropy_von_neumann`() that takes as arguments the MPO and the MPO site to orthogonalise about [26].

```
function entropy_von_neumann(psi::MPO, b::Int)
    s = siteinds(psi)
    orthogonalize!(psi, b)
    _,S = svd(psi[b], (linkind(psi, b-1), s[b]))
    sum = 0.0
    SvN = 0.0
    for n in 1:dim(S, 1)
      p = S[n,n]^2
      println(S[n,n])
      sum += p
      SvN -= p * log(p)
    end
    @show sum
    return SvN
  end
```

This function performs SVD via the function `svd()` about the orthogonality center MPO site `psi[b]` to return the matrix of singular values `S`. By iterating through the singular values `S[n,n]`, the von Neumann entropy can be calculated according to Eq. 2.31.

For the Grover MPO, this was calculated for an orthogonality center site `b` using

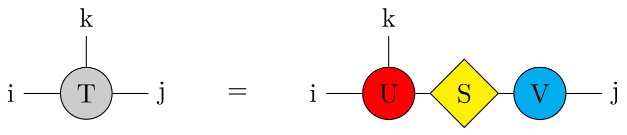
```
SvN = entropy_von_neumann(MPO_TOT, b)
@show SvN
```

SVD in ITensor

Recalling the SVD method from Section 2.5.5, `ITensor` implements this method for a tensor stored as an `ITensor` object using the function `svd()`. Labelling the ITensor's indices as `i,j,k` to then specify some indices (eg: `i,k`) in the argument of the `svd()` function allows the tensor to be reshaped as a matrix, where the specified indices become the "row" indices, and those excluded become "column" indices [26].

```
i = Index(3,"i")
j = Index(4,"j")
k = Index(5,"k")

T = ITensor(i,j,k)
U, S, V = svd(T, (i,k))

@show norm(U*S*V-T)
```

`ITensor`'s SVD operation can be illustrated in the diagram [26]:



The truncated SVD method may be called by inclusion of the `cutoff` argument in the `svd()` function. By fixing the `cutoff` value to any real number ϵ, the SVD method will discard singular values σ_n such that the truncation error is less than ϵ,

$$\frac{\sum_{n\in\text{discarded}}\sigma_n^2}{\sum_n \sigma_n^2} < \epsilon \tag{3.1}$$

Alternatively, inclusion of the `maxdim` argument, assigned some integer M, in the `svd()` function ensures only the largest M singular values are kept during the SVD.

3.3 Implementation: Python

The machinery behind converting tensors to MPOs and contracting MPOs with each other was in-built to `ITensor` in `Julia`. This machinery, notably SVD decomposition and the zip-up algorithm, was programmed explicitly in `Python` for greater understanding of the simulation being performed.

Like the `Julia` implementation, the `Python` implementation constructs an Oracle MPO and a Diffuser MPO for easy repetition later in the circuit. Unlike the `Julia` implementation, different contraction orders are employed for more efficient contractions for use in the zip-up algorithm.

Order-3 and order-4 tensors that make up MPSs and MPOs are created in using `NumPy` arrays. Knowing which elements to populate the arrays with to encode a gate layer was achieved using the finite state automata method to construct MPSs and MPOs. This method is detailed in Appendix B, and was key to implementing the multi-qubit controlled-Z and controlled-X gates necessary in the Oracle MPO and Diffuser MPO respectively. Once these MPOs were constructed, contractions were carried out via the zip-up algorithm.

Zip-Up Algorithm

In this approach, the initial qubit state MPS (rank χ) is contracted with the adjacent MPO (rank k) to generate a new MPS [27]. See Appendix C for details on the algorithm itself. For simulating Grover's algorithm, this process repeats for the newly adjacent MPO, until a final MPS is generated. This encodes the action of the Grover MPO on the initial state.

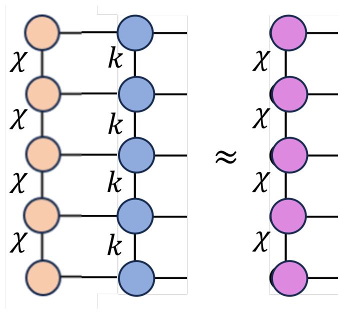


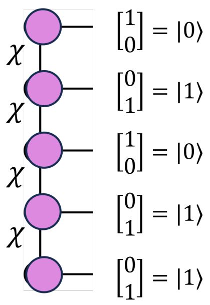
Figure 3.1: The action of the zip-up algorithm on an MPS and MPO to produce a new MPS.

To achieve contraction over indices between two tensors, the `numpy.tensordot()` function was used, taking both tensors and the index variables as arguments. The SVD of a tensor utilised the `numpy.linalg.svd()` function that took the flattened tensor as an argument and returned three `NumPy` arrays. Truncation was then performed using a for-loop to remove singular values less than the specified `cutoff` value.

The runtime of this algorithm scales as $N(\chi^3kd^2 + \chi^2k^2d^2)$ [27] where N is the number of MPS sites and d is the dimension of the 'real' indices. For Grover's algorithm, $\chi = k = d = 2$, since the largest bond dimension MPOs are constructed from the multi-qubit controlled-U gates, which have bond dimension at most two [15].

The zip-up algorithm is implemented using the `apply_diffuser_2( MPS_list, N, cutoff )` function. The `MPS_list` list stores all the tensors for Grover's algorithm, and replaces each tensor with its MPS computed from the zip-up algorithm.

All 'real' indices of final MPS can be contracted with vectors for the $|0\rangle$ state or $|1\rangle$ state in the order of a desired measurement bit string. For example, to measure the computational basis state $|01011\rangle$, the final MPs is contracted as



χ
$\begin{bmatrix}1\\0\end{bmatrix} = |0\rangle$
$\begin{bmatrix}0\\1\end{bmatrix} = |1\rangle$
$\begin{bmatrix}1\\0\end{bmatrix} = |0\rangle$
$\begin{bmatrix}0\\1\end{bmatrix} = |1\rangle$
$\begin{bmatrix}0\\1\end{bmatrix} = |1\rangle$

This will yield a scalar value, which will equal the probability amplitude of Grover's algorithm outputting that computational basis state.

Chapter 4

Results and Discussion

4.1 Results from Julia Implementation

Bond Dimension Results

Data was collected for simulations of the n-qubit Grover's algorithm as an MPO when varying the number of qubits n and the SVD cutoff value ε. The bond dimensions for each bond in the Grover MPO were outputted and recorded, from which the maximum bond dimensions for a given MPO size could be found.

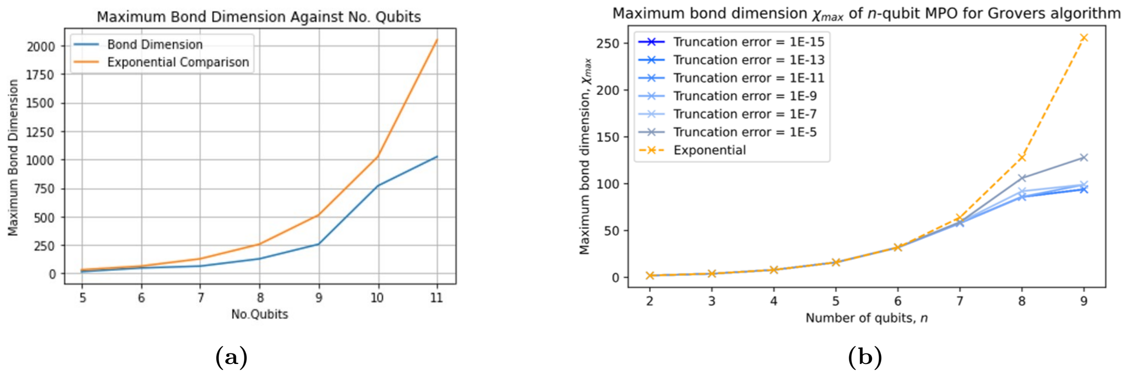


Figure 4.1: The maximum bond dimension $\chi_{\max}$ observed across the Grover MPO with (a) no cutoff during SVD, and (b) cutoffs during SVD, using the `Julia` implementation.

In Figure 4.1(a), maximum bond dimension $\chi_{\max}$ across the Grover MPO was observed to increase exponentially but with a slower growth factor than 2^n. This is most evident when observing the maximum bond dimensions for higher qubit numbers $n = 10$ and $n = 11$. In Figure 4.1(b), maximum bond dimension $\chi_{\max}$ for the truncated Grover MPO construction with varying SVD cutoff value ε, ranging from the machine precision 10^{-16} to 10^{-6}, was observed to grow with a factor closer to 2^n than with no cutoff value. The largest diversions from this growth occurred at larger qubit numbers, around $n = 8$ and $n = 9$. Values of the maximum bond dimension plateaued around $\chi_{\max} \approx 100$ for the truncated MPO up to $n = 9$ qubits, versus steady growing $\chi_{\max} \approx 250$ for the exact MPO at $n = 9$ qubits, with maximum bond dimension being observed to reach $\chi_{\max} \approx 1000$ for the exact MPO at $n = 11$ qubits.

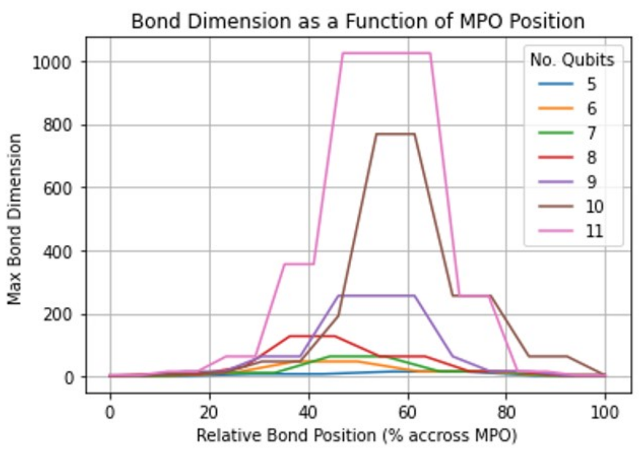


Figure 4.2: Bond dimension χ values for each bond in the n qubit Grover MPO for varying n.

The maximum bond dimension for a given Grover MPO was found to be concentrated around central MPO bonds, as shown in Figure 4.2 for the exact MPO case. For all MPO sizes tested, the first bond dimension plateaued at $\chi_1 = 4$, and the final bond dimension plateaued at $\chi_{n-1} = 2$. On average, the top half of bonds in the Grover MPO had larger bond dimension values than the lower half for all number of qubits tested.

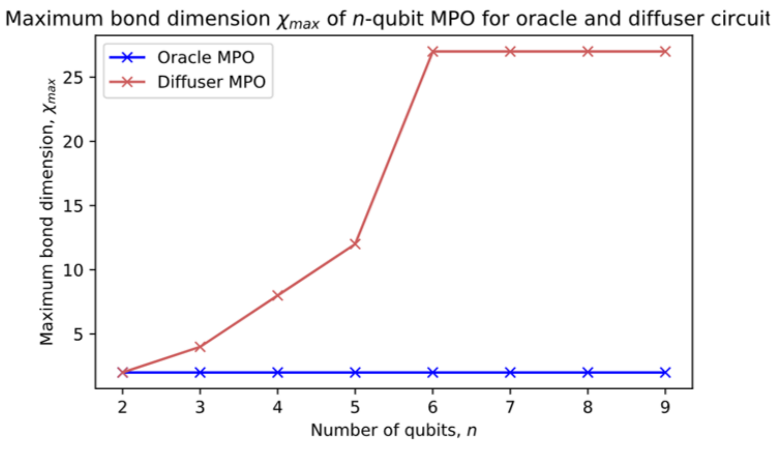


Figure 4.3: Caption

Maximum bond dimensions for the Oracle MPO and the Diffuser MPO were observed varying the number of qubits. Both MPOs were expected to remain at maximum bond dimension $\chi_{\max} = 2$ due to the multi-qubit control gates they each contain [15]. The Oracle MPO was found to behave as expected, with maximum bond dimension remaining at $\chi_{\max} = 2$ for all observed number of qubits. The Diffuser MPO initially grew from $\chi_{\max} = 2$ to $\chi_{\max} = 28$ after $n = 6$ qubits where it plateaued for all greater number of qubits. This unexpected behaviour suggests a faulty implementation of the Diffuser MPO, most likely the multi-qubit control gate, although the reason for the fault remains unknown.

Singular Values Results

Singular values found via SVD at a particular MPO site for the Grover MPO were found to approach infinity for their largest values, with the majority of values being greater than one. This may have been a contributing factor to the `Julia` code failing to produce bond dimension values other than

$\chi = 1$ for number of qubits greater than $n = 11$. This fault significantly impacted research into approximating Grover's algorithm as an MPO, limiting data collection to small numbers of qubits, where the technique should have lent itself to simulating up to $n \approx 30$ qubits.

State Selection Results

From Section 2.4.1, the outcome of Grover's algorithm should be a vector of probability amplitudes that when squared correspond to the probability of the circuit outputting that computational basis state. The computational basis state with the largest probability amplitude should correspond to the state selected by the Oracle implemented.

The probability amplitudes returned via contracting the Grover MPO with the initial MPS and the intended final state according to the implemented oracle were found to be close to zero, not as intended. This was found to be the case for various oracles. The reason for this fault remains unknown.

4.2 Results from Python Implementation

Bond Dimension and State Selection Results

Similarly to the `Julia` implementation, data was collected for simulations of the n-qubit Grover's algorithm as an MPO when varying the number of qubits n and the SVD cutoff value ε. The bond dimensions for each bond in the Grover MPO were outputted and recorded, from which the maximum bond dimensions for a given MPO size could be found.

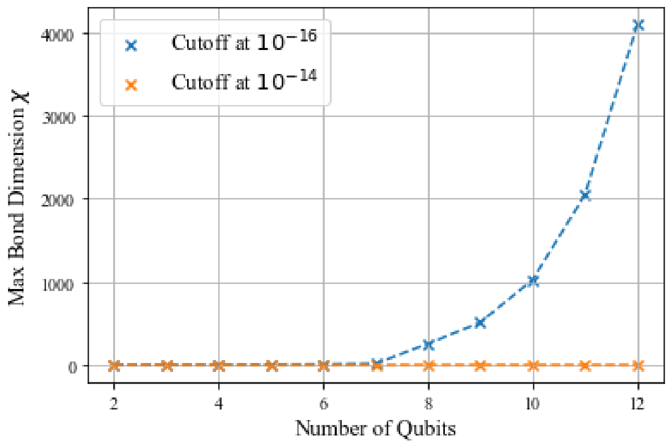


Figure 4.4: The maximum bond dimension $\chi_{\max}$ observed across the Grover MPO with no cutoff during SVD using the `Python` implementation.

The maximum bond dimension $\chi_{\max}$ was observed to increase exponentially with the number of qubits n for a cutoff value equal to the machine precision 10^{-16}. Increasing the cutoff value to 10^{-15} resulted in the maximum bond dimension plateauing at $\chi_{\max} = 3$, with further increases to cutoff value of 10^{-14} and above yielding constant maximum bond dimension of $\chi_{\max} = 2$ for all tested number of qubits. This behaviour is illustrated in Figure 4.4. In all simulations, all bonds but the final bond remained at a bond dimension of χ_2 regardless of the number of qubits n. Only the final bond was observed to change depending on the cutoff value, and thus dictated $chi_{\max}$.

This result aligns with expectations for using the zip-up algorithm with MPOs of maximum bond dimension $\chi_{\max} = 2$ from the multi-qubit control gates. Simulations of Grover's algorithm approximated as an MPO were ran for up to 35 qubits, with $\chi_{\max} = 2$ results yielding $O(1)$ probability amplitudes for the desired computational basis state from the specific oracle implemented, whilst all other probability amplitudes were negligible. The average probability amplitude of the desired computational basis state for tests of all number of qubits from $n = 2$ to $n = 30$ was 0.995 ± 0.014. Hence, the `Python` implementation successfully simulated Grover's algorithm as an approximate MPO with constant maximum bond dimension $\chi_{\max} = 2$ with number of qubits n for cutoff values 10^{-14} and above. This implies that Grover's algorithm can be approximated in a linear number of parameters $\mathcal{O}(2n \cdot 2^2) = \mathcal{O}(8n)$ with the number of qubits, providing exponential saving over the state vector's $\mathcal{O}(2^n)$ necessary parameters.

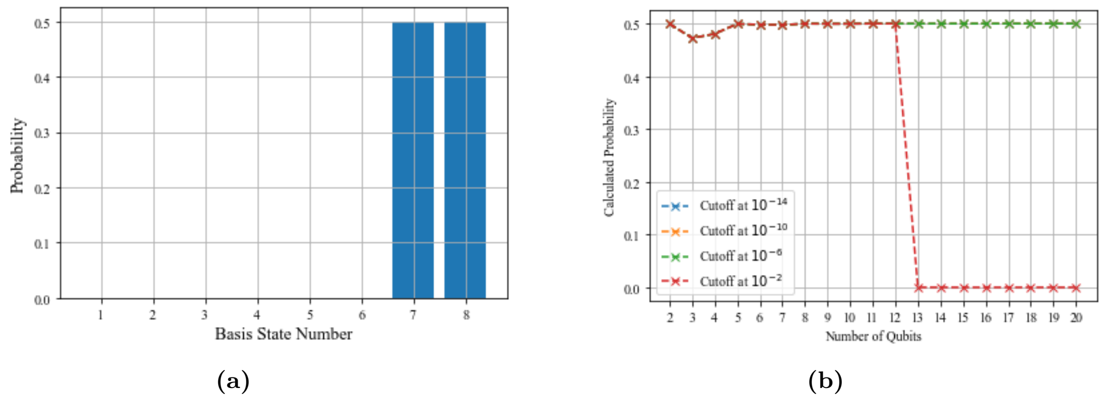


Figure 4.5: (a) The outcome of running a 2 qubit Grover's algorithm simulation with a third qubit for the oracle in `Python`. Although it looks like two outcomes are equally likely, these represent the states $|110\rangle$ and $|111\rangle$, hence the element '11' from the first two qubits is outputted with certainty. (b) A plot of probabilities for outputting the desired state as a function of the number of qubits and the cutoff values.

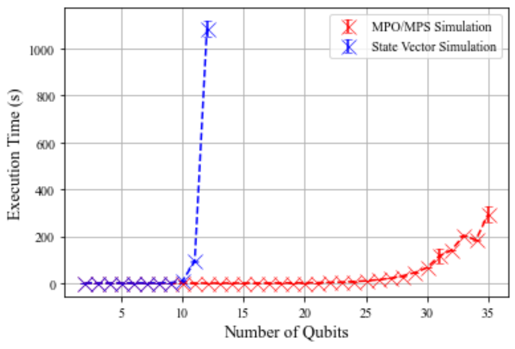
An example of the outputted probability when the final MPS was contracted with the desired final state is shown in Figure 4.5(a). Since the final qubit is a dummy qubit for encoding database elements, the two outcomes $|110\rangle$ and $|111\rangle$ correspond to the same element '11'. This was observed for different numbers of qubits, and the results were plotted in Figure 4.5(b). An unexpected dip in probability occurs for small qubit numbers around $n = 3$ and $n = 4$, but returns to high probability for all other qubit numbers. For a large cutoff value of 10^{-2}, the probability of obtaining the desired state becomes negligible after $n = 13$ qubits, since the final bond dimension drops to $\chi = 1$ and not enough information of the system is retained for good approximation of the Grover's algorithm.

Timing Results

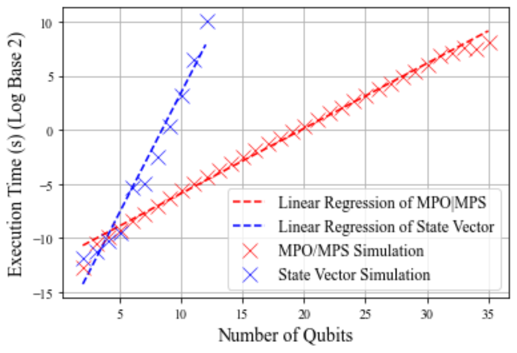
Given the high probability of successfully outputting the desired computational basis state, timings of the Grover MPO simulation were recorded versus the timings of a state vector simulation for up to $n = 35$ qubits. For each simulation, three timing measurements were taken, and their average plotted with their standard deviation as the error. It can be seen from Figure 4.6(a) that the state vector simulation became intractable after $n = 12$ qubits, whereas the MPO simulation had much slower growth rate in execution time. The same time taken to simulate $n = 11$ qubits in the state vector simulation could have simulated $n \approx 30$ qubits using the MPO approach, whilst retaining high probability of success for the simulated Grover's algorithm. Simulation of $n = 35$ qubits took approximately 350 seconds when run on a single core on an Apple M1 processor.

When plotted on a logarithmic axis, as in Figure 4.6(b), it is clear that both simulations grow exponentially but with significantly different growth factors. Using linear regression, it was found that the execution time of the state vector simulation had growth factor $\lambda = 0.212$, growing as $2^{2.212n}$ with number of qubits n; whereas the execution time of the MPO simulation had growth factor $\lambda = 0.600$, growing as $2^{0.600n}$ with number of qubits n.

The speedups using the MPO simulation over the state vector simulation were calculated and graphed in Figure 4.7(a) . The speedup appears to grow exponentially in the number of qubits n, thus achieving a greater than constant factor speedup over the state vector simulation. To contextualise this speedup, Figure 4.7(b) displays a logarithmic plot of execution time with number of qubits for a graph with the same growth rate of the MPO simulation, but projected for higher qubit numbers. Simulations of $n = 40$ qubits are projected to have an execution time on the order of hours, $n = 60$ qubits on the order of days, and $n = 80$ qubits on the order of years.

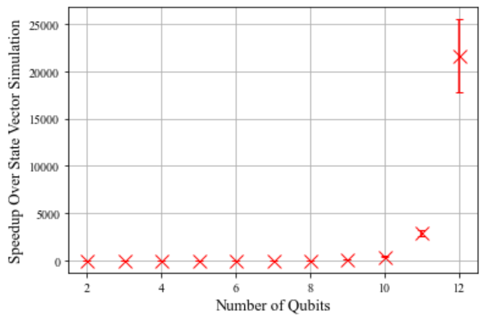


(a)

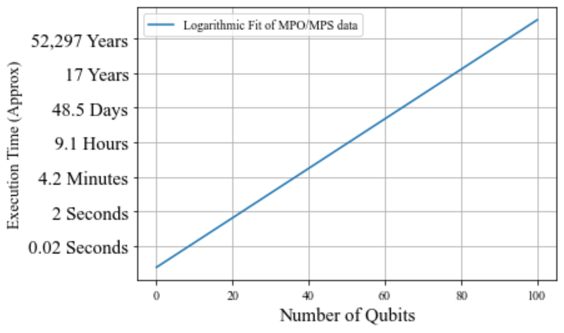


(b)

Figure 4.6: Execution time comparison for basic state vector simulation of Grover's algorithm and the approximate MPO simulation against the number of qubits simulated. Execution time axis plotted normally in (a) and logarithmically in (b).



(a)



(b)

Figure 4.7: (a) Speedup of the MPO simulation over the state vector simulation as a function of the number of qubits. (b) The scaling of execution time of the MPO simulation projected to $n = 100$ qubits.

Singular Value Results

Singular values for each MPO site were recorded. For simulations with cutoff value above 10^{-14}, all bond dimensions had values $\chi = 2$, and therefore contained 2 singular values of $\sigma_1 = 1.414$ and $\sigma_2 = 1$. Although these values do not exhibit faster than exponential decay between them, two data points is far too few to draw conclusions on their decay rate.

4.3 Discussion

The `Python` implementation results of constant maximum bond dimension $\chi_{\max} = 2$ suggest that the MPO representation can well approximate Grover's algorithm for number of qubits up to $n = 30$

and SVD cutoff values of 10^{-14} or above. This would suggest that Grover's algorithm does not have as large entanglement generating power as previously believed, according to the von Neumann entropy based off its two singular values. Since truly quantum advantaged algorithms would have an entanglement generating power large enough to not be simulable classically for large qubit systems (in other words, their MPO representations have maximum bond dimensions that diverge with number of qubits to maintain a given approximation error), these results suggest that Grover's algorithm may not be as quantum advantaged as previously believed. This would imply that the some of quantum advantage in algorithms utilising Grover's algorithm as a subroutine stems from structures other than Grover's algorithm.

Chapter 5

Conclusion and Outlook

Grover's algorithm was approximated as an MPO and simulated in two different implementations. It was found that the MPO representation can well approximate Grover's algorithm for number of qubits up to $n = 30$ and SVD cutoff values of 10^{-14} or above. The `Julia` implementation of the Grover MPO showed exponentially increasing maximum bond dimension $\chi_{\max}$, but suffered faults regarding the maximum bond dimensions in the Diffuser MPO and returning the correct probability amplitudes. Research on the growth maximum bond dimension $\chi_{\max}$ with number of qubits n was limited to $n = 11$ qubits, where results up to $n \approx 30$ were targeted.

Singular values of Grover MPO sites were found to contain larger than expected values, preventing reliable entanglement calculations using the von Neumann entropy. Were these values to have behaved as expected (taken values between 0 and 1), analysis of their decay rate could have occurred, with faster than exponential decay indicating simulability of Grover's algorithm via MPO.

The `Python` implementation of the Grover MPO showed constant maximum bond dimension $\chi_{\max}$ for all n qubit simulations up to $n = 35$ using cutoff values of 10^{-14} or above. The correct probability amplitudes were outputted, with $\mathcal{O}(1)$ probability only for the desired computational basis state from the oracle simulated.

Run time analysis of the `Python` implementation suggest a growth factor of $\lambda = 0.600$ for an execution time that scales as $2^{\lambda n} = 2^{0.600n}$. This significantly improved on the execution time of the state vector simulation of Grover's algorithm, with speedup observed to be exponential in the number of qubits.

5.1 Future Research Directions

The implementation in this project determined bond dimension as a function of SVD cutoff value. However, the fidelity could also be investigated for Grover's algorithm, determining cutoff value as a function of bond dimension.

Since the aim of this project was to implement Grover's algorithm as an approximate MPO for the purposes of investigating bond dimension, little attention was paid to the contraction orders used in generating our final MPO. Future research could look for faster simulation of Grover's algorithm as an approximate MPO by optimising contraction order of indices for the MPOs used in this algorithm [28, 29, 30]. Optimal contraction orders may also be possible for simulation on distributed classical systems, with parallel implementation of the zip-up algorithm allowing for faster simulation.

Whilst the Grover's algorithm implemented in this project only searched for one element, the algorithm can be generalised for multiple elements listed in a set $\mathcal{S}$. Modifying this project's implementations to yield an MPO approximating a multiple element Grover's algorithm would allow for further investigation as to whether maximum bond dimension $\chi_{\max} = 2$ remains constant. Expanding on this, the concepts underpinning Grover's algorithm can be generalised to another algorithm known as quantum amplitude amplification [31]. In this algorithm, there are multiple desired elements in set $\mathcal{S}$ and access to an oracle as in Grover's algorithm, but now also access to a "guessing" algorithm $\mathcal{A}$ that produces potential solutions. Since $\mathcal{A}$ applies a map as

$$\mathcal{A}|0^{\otimes n}\rangle = \sum_{x\in\{0,1\}^n} \gamma_x |x\rangle,$$

for some coefficients $\{\gamma_x\}$, the probability a desired solution is obtained is then

$$p = \sum_{x\in\mathcal{S}} |\gamma_x|^2.$$

Here, $\mathcal{A}$ takes on the role that the Hadamard gate H fulfills in Grover's algorithm. Hence, an interesting comparison could be drawn between the results of the multiple element Grover's algorithm as an MPO and quantum amplitude amplification as an MPO, and if their maximum bond dimensions χ_{max} grew with number of qubits identically.

The implementation in this project focused on converting each gate layer in the Grover's algorithm quantum circuit to an MPO. Using the method of finite state automata [32] (see Appendix B) could allow for an alternative construction of Grover's algorithm as an MPO just from the O and D operators listed in Eqns 2.13 and 2.14 respectively. Observing whether this construction produced corroborating results to the `Python` implementation in this project would be of great interest. This approach could also be applied to all suggestions in the previous paragraph.

Bibliography

[1] J. I. Cirac, D. Pérez-García, N. Schuch, and F. Verstraete, "Matrix product states and projected entangled pair states: Concepts, symmetries, theorems," *Reviews of Modern Physics*, vol. 93, Dec. 2021.

[2] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information.* Cambridge University Press, 2000.

[3] D. Deutsch and R. Jozsa, "Rapid solution of problems by quantum computation," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 439, pp. 553 – 558, 1992.

[4] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Journal on Computing*, vol. 26, p. 1484–1509, Oct. 1997.

[5] L. K. Grover, "A fast quantum mechanical algorithm for database search," 1996.

[6] A. M. Dalzell, S. McArdle, M. Berta, P. Bienias, C.-F. Chen, A. Gilyén, C. T. Hann, M. J. Kastoryano, E. T. Khabiboulline, A. Kubica, G. Salton, S. Wang, and F. G. S. L. Brandão, "Quantum algorithms: A survey of applications and end-to-end complexities," 2023.

[7] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. S. L. Brandao, D. A. Buell, B. Burkett, Y. Chen, Z. Chen, B. Chiaro, R. Collins, W. Courtney, A. Dunsworth, E. Farhi, B. Foxen, A. Fowler, C. Gidney, M. Giustina, R. Graff, K. Guerin, S. Habegger, M. P. Harrigan, M. J. Hartmann, A. Ho, M. Hoffmann, T. Huang, T. S. Humble, S. V. Isakov, E. Jeffrey, Z. Jiang, D. Kafri, K. Kechedzhi, J. Kelly, P. V. Klimov, S. Knysh, A. Korotkov, F. Kostritsa, D. Landhuis, M. Lindmark, E. Lucero, D. Lyakh, S. Mandrà, J. R. McClean, M. McEwen, A. Megrant, X. Mi, K. Michielsen, M. Mohseni, J. Mutus, O. Naaman, M. Neeley, C. Neill, M. Y. Niu, E. Ostby, A. Petukhov, J. C. Platt, C. Quintana, E. G. Rieffel, P. Roushan, N. C. Rubin, D. Sank, K. J. Satzinger, V. Smelyanskiy, K. J. Sung, M. D. Trevithick, A. Vainsencher, B. Villalonga, T. White, Z. J. Yao, P. Yeh, A. Zalcman, H. Neven, and J. M. Martinis, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, pp. 505–510, Oct. 2019.

[8] J. Preskill, "Quantum computing in the nisq era and beyond," *Quantum*, vol. 2, p. 79, Aug. 2018.

[9] I. L. Markov and Y. Shi, "Simulating quantum computation by contracting tensor networks," *SIAM Journal on Computing*, vol. 38, p. 963–981, Jan. 2008.

[10] I. V. Oseledets, "Tensor-train decomposition," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2295–2317, 2011.

[11] G. Vidal, "Efficient classical simulation of slightly entangled quantum computations," *Phys. Rev. Lett.*, vol. 91, p. 147902, Oct 2003.

[12] J. Chen, E. Stoudenmire, and S. R. White, "Quantum fourier transform has small entanglement," *PRX Quantum*, vol. 4, Oct. 2023.

[13] N. Yoran and A. J. Short, “Efficient classical simulation of the approximate quantum fourier transform,” *Physical Review A*, vol. 76, Oct. 2007.

[14] D. Aharonov, Z. Landau, and J. Makowsky, “The quantum fft can be classically simulated,” 2007.

[15] P. Gelß, S. Klus, S. Knebel, Z. Shakibaei, and S. Pokutta, “Low-rank tensor decompositions of quantum circuits,” 2023.

[16] I. V. Oseledets and E. E. Tyrtyshnikov, “Breaking the curse of dimensionality, or how to use svd in many dimensions,” *SIAM Journal on Scientific Computing*, vol. 31, no. 5, pp. 3744–3759, 2009.

[17] W. Hackbusch and S. Kühn, “A new scheme for the tensor representation,” *Journal of Fourier Analysis and Applications*, vol. 15, pp. 706–722, Oct. 2009.

[18] L. Grasedyck, “Hierarchical singular value decomposition of tensors,” *SIAM Journal on Matrix Analysis and Applications*, vol. 31, no. 4, pp. 2029–2054, 2010.

[19] J. D. Carroll and J.-J. Chang, “Analysis of individual differences in multidimensional scaling via an n-way generalization of “Eckart-Young” decomposition,” *Psychometrika*, vol. 35, pp. 283–319, Sept. 1970.

[20] L. De Lathauwer, B. De Moor, and J. Vandewalle, “A multilinear singular value decomposition,” *SIAM Journal on Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1253–1278, 2000.

[21] “Indexing by latent semantic analysis,” *Journal of the Association for Information Science and Technology*, vol. 41, no. 6, pp. 391–407, 1990.

[22] L. R. Tucker, “Some mathematical notes on three-mode factor analysis,” *Psychometrika*, vol. 31, pp. 279–311, Sept. 1966.

[23] C. H. Bennett, D. P. DiVincenzo, J. A. Smolin, and W. K. Wootters, “Mixed-state entanglement and quantum error correction,” *Physical Review A*, vol. 54, p. 3824–3851, Nov. 1996.

[24] N. Schuch, M. M. Wolf, F. Verstraete, and J. I. Cirac, “Entropy scaling and simulability by matrix product states,” *Physical Review Letters*, vol. 100, Jan. 2008.

[25] C. R. Wie, “Phase factors in singular value decomposition and schmidt decomposition,” 2022.

[26] M. Fishman, S. R. White, and E. M. Stoudenmire, “Codebase release 0.3 for ITensor,” *SciPost Phys. Codebases*, pp. 4–r0.3, 2022.

[27] E. M. Stoudenmire and S. R. White, “Minimally entangled typical thermal state algorithms,” *New Journal of Physics*, vol. 12, p. 055026, may 2010.

[28] C. Ibrahim, D. Lykov, Z. He, Y. Alexeev, and I. Safro, “Constructing optimal contraction trees for tensor network quantum circuit simulation,” 2022.

[29] B. O’Gorman, “Parameterization of tensor network contraction,” 2019.

[30] J. Gray and S. Kourtis, “Hyper-optimized tensor network contraction,” *Quantum*, vol. 5, p. 410, Mar. 2021.

[31] G. Brassard, P. Høyer, M. Mosca, and A. Tapp, “Quantum amplitude amplification and estimation,” 2002.

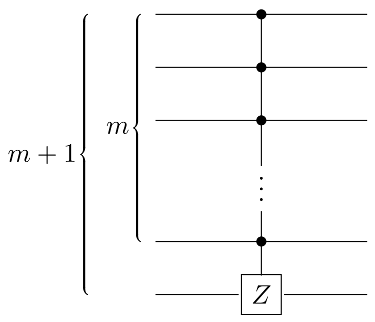
[32] G. M. Crosswhite and D. Bacon, “Finite automata for caching in matrix product algorithms,” *Physical Review A*, vol. 78, July 2008.

Appendix A

Deriving matrices used in Julia code

A.1 Multi-qubit controlled-Z Gate

For an $m+1$ qubit controlled-Z gate $\mathrm{C}^{m+1}Z$ represented by the diagram



the associated $(2^{m+1} \times 2^{m+1})$ matrix can be constructed via the relation

$$\mathrm{C}^m Z = (\underbrace{P_0 \otimes ... \otimes P_0}_{m} \otimes I) + (\underbrace{P_1 \otimes ... \otimes P_1}_{m} \otimes Z), \tag{A.1}$$

where $Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$, $P_0 = |0\rangle\langle 0| = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$ is the projector onto the $|0\rangle$ basis and $P_1 = |1\rangle\langle 1| = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$ is the projector onto the $|1\rangle$ basis.

Carrying out the tensor products of matrices, Eq. A.3 can be written as

$$\mathrm{C}^m Z = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & & 0 & & \\ & & & & \ddots & \\ & & & & & 0 \end{bmatrix} + \begin{bmatrix} 0 & & & & \\ & \ddots & & & \\ & & 0 & & \\ & & & 1 & & \\ & & & & \ddots & \\ & & & & & -1 \end{bmatrix} \tag{A.2}$$

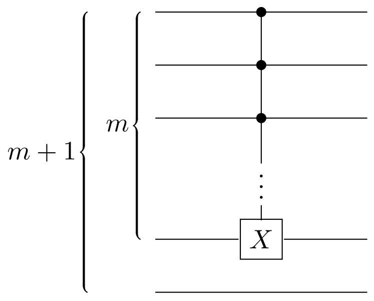
$$= \begin{bmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & & 1 & & \\ & & & & \ddots & \\ & & & & & -1 \end{bmatrix}$$

Therefore, this is effectively the $(2^{m+1} \times 2^{m+1})$ identity matrix with the last diagonal element changed to -1. Hence, in `Julia`, this matrix for N qubits is created using

```
    M = Matrix(I, 2^N, 2^N)
    M = Int.(M)
    M[2^N, 2^N] = -1
```

A.2 Multi-qubit controlled-X Gate

When utilised in Grover's algorithm, the m qubit controlled-X gate $\mathrm{C}^m X$ gate is used with the $(m+1)$-th qubit uninvolved, as represented in the diagram:



The associated $(2^{m+1} \times 2^{m+1})$ matrix can be constructed via the relation

$$\mathrm{C}^{m-1}X \otimes I = \left[(\underbrace{P_0 \otimes ... \otimes P_0}_{m-1} \otimes I) + (\underbrace{P_1 \otimes ... \otimes P_1}_{m-1} \otimes X)\right] \otimes I, \tag{A.3}$$

where $X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$, $P_0 = |0\rangle\langle 0| = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$ is the projector onto the $|0\rangle$ basis and $P_1 = |1\rangle\langle 1| = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$ is the projector onto the $|1\rangle$ basis. This can be rewritten [15] as

$$\mathrm{C}^{m-1}X \otimes I = \left[(I^{\otimes m}) + (\underbrace{P_1 \otimes ... \otimes P_1}_{m-1} \otimes (X - I))\right] \otimes I \tag{A.4}$$

where $X - I = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix}$. The term in square brackets can be calculated as

$$\begin{bmatrix} 1 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & 1 \end{bmatrix} + \begin{bmatrix} 0 & & & \\ & \ddots & & \\ & & -1 & 1 \\ & & 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & & & \\ & \ddots & & \\ & & 0 & 1 \\ & & 1 & 0 \end{bmatrix}$$

(A.5)

This matrix is then tensor producted with a (2×2) identity matrix I to give

$$\begin{bmatrix} 1 & & & \\ & \ddots & & \\ & & 0 & 1 \\ & & 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & \ddots & & & \\ & & & 0 & 0 & 1 & 0 \\ & & & 0 & 0 & 0 & 1 \\ & & & 1 & 0 & 0 & 0 \\ & & & 0 & 1 & 0 & 0 \end{bmatrix}. \tag{A.6}$$

This final matrix is implemented in `Julia` for N qubits as

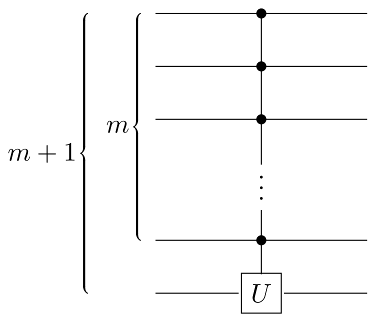
```julia
M2 = Matrix(I, 2^N, 2^N)
M2 = Int.(M2)

if N==2
    M2[2^N, 2^N] = 0
    M2[2^N-1, 2^N-1] = 0

    M2[2^N, 2^N-1] = 1
    M2[2^N-1, 2^N] = 1
else

    M2[2^N, 2^N] = 0
    M2[2^N-1, 2^N-1] = 0
    M2[2^N-2, 2^N-2] = 0
    M2[2^N-3, 2^N-3] = 0

    M2[2^N, 2^N-2] = 1
    M2[2^N-2, 2^N] = 1

    M2[2^N-1, 2^N-3] = 1
    M2[2^N-3, 2^N-1] = 1
end
```

Appendix B

Deriving MPO tensors for multiple qubit controlled-U gates via weighted finite state automata

B.1 (m+1)-qubit Controlled-U Gates

A quantum register of n qubits can be acted on by an $(m+1)$ qubit gate $\mathrm{C}^m U$, consisting of m control qubits and one target qubit, for some unitary U.



Typically, the $\mathrm{C}^m U$ gate is written as

$$\mathrm{C}^m U = (\underbrace{P_0 \otimes ... \otimes P_0}_{m} \otimes I) + (\underbrace{P_1 \otimes ... \otimes P_1}_{m} \otimes U) \tag{B.1}$$

where $P_0 = |0\rangle\langle 0| = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$ is the projector onto the $|0\rangle$ basis and $P_1 = |1\rangle\langle 1| = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$ is the projector onto the $|1\rangle$ basis. However, as proved in Ref. [15], this may also be written as

$$\mathrm{C}^m U = I^{\otimes m+1} + (\underbrace{P_1 \otimes ... \otimes P_1}_{m} \otimes (U - I)). \tag{B.2}$$

To implement these gates for the purposes of simulating Grover's algorithm in `Julia` and `Python`, an MPO must be constructed from the $\mathrm{C}^m U$ gate. Finite state automata method allows construction of this MPO by considering the individual terms summed in the operator.

B.2 Finite State Automata

A finite state automaton models a machine that moves from one state to another based on an input signal [32]. In a given state, the machine can have multiple possible transitions to the next state, with each transition weighted by a number. From an initial state to a final state, multiple possible paths may exist with a weight equal to the product of weights along the path. The automaton then outputs the sum of all weights of all paths from all initial to all final states [32].

A matrix product operator can be viewed as a specific instance of a weighted finite state automaton, with each node being a state and each edge being a transition. A transition matrix can be constructed for each point in the string.

Lemma B.2.1. *(From Ref. [32]) Just as it is possible to view a matrix product state as a special case of a weighted finite state automaton, it is always possible to construct a matrix product state from a weighted finite automaton.*

Given Lemma B.2.1, a procedure to construct an MPO can be given as:

1. Construct a weighted finite state automata that generates the pattern of the operator,
2. Translate this into a matrix product operator diagram,
3. Determine matrices based on this diagram.

The C^mU operator in Eq. B.2 contains two terms. Hence, one path of the automata will produce the first term $I^{\otimes m+1}$, and the only other path will produce $P_1 \otimes ... \otimes P_1 \otimes (U - I)$, where both paths share starting and finishing states respectively. This diagram is visualised in Figure B.1.

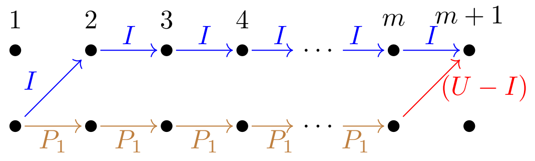


Figure B.1: Matrix Product Operator diagram for C^mU operator constructed from a weighted finite state automata.

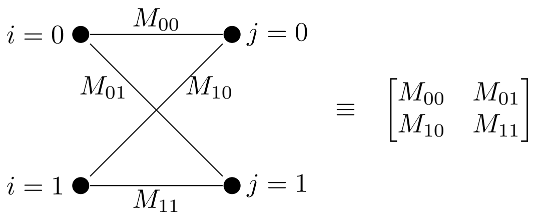
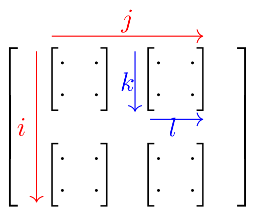

$i=0$
$i=1$
$j=0$
$j=1$
M_{00}
M_{01}
M_{10}
M_{11}
$\equiv \begin{bmatrix} M_{00} & M_{01} \\ M_{10} & M_{11} \end{bmatrix}$

Figure B.2: Mapping of diagram edges to matrix elements of a transition matrix.

The mapping of diagram edges to matrix elements is illustrated in Figure B.2. Using this, the matrix product operator diagram for C^mU can be translated to matrices as:

$$C^mU = \begin{bmatrix} I & P_1 \end{bmatrix} \underbrace{\begin{bmatrix} I & \\ & P_1 \end{bmatrix} \cdots \begin{bmatrix} I & \\ & P_1 \end{bmatrix}}_{m-1} \begin{bmatrix} I \\ (U-I) \end{bmatrix} \tag{B.3}$$

However, this representation involves matrices of matrices, since I and P_1 are both 2×2 matrices. Hence, an element in the first or final vector is specified by three indices, whereas an element in any of the $m-1$ matrices is specified by four indices, as illustrated below.
Hence, Eq. B.3 presents an order-3 tensor contracted over one index with an order-4 tensor, again contracted over one index with an order-4 tensor, etc., and finally contracted over one index with



an order-3 tensor. Therefore, Eq. B.3 encodes the $\mathrm{C}^m U$ operator as an MPO.

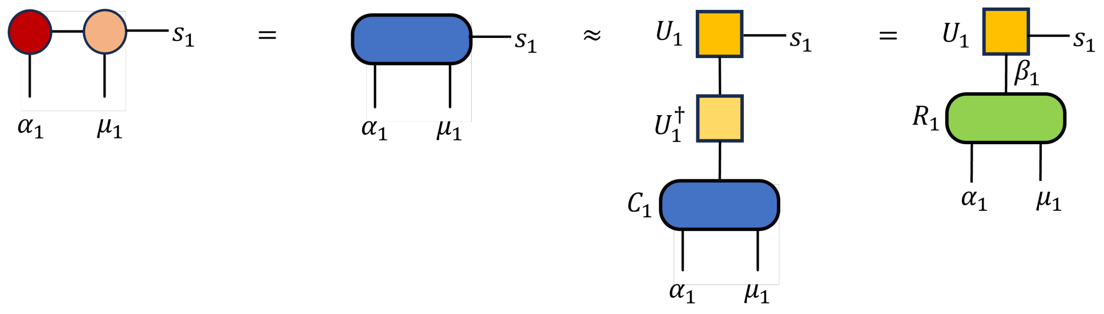
Note that the dimension of the contracted indices corresponds to the bond dimension of the virtual indices in an MPO. Here, the dimension of each index is 2, since a (2×2) matrix was sufficient to encode all paths in the MPO diagram, resulting in a maximum bond dimension $\chi_{max} = 2$ sufficient to encode a $\mathrm{C}^m U$ gate for any m and any unitary matrix U. This result is reiterated in Ref. [15].

Appendix C

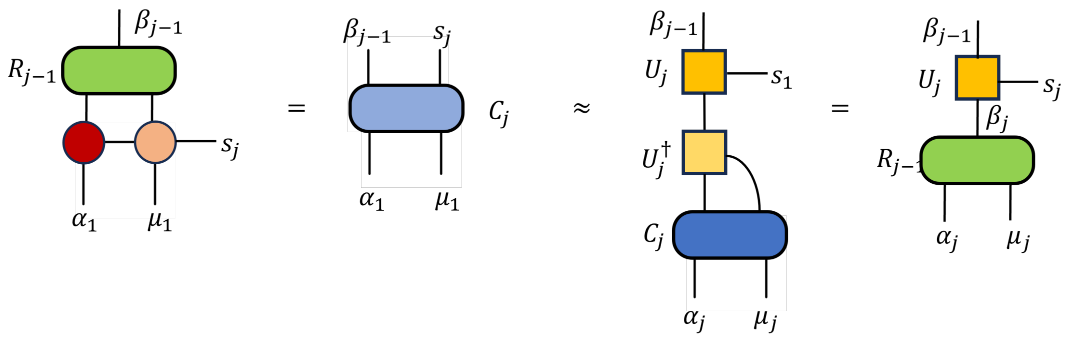
The Zip-Up Algorithm

The zip-up algorithm contracts an MPS and an MPO with the same number of sites together to produce another MPS [27].

First, it contracts the first tensor in the MPS and MPO together to form tensor $\mathbf{C}_1$ which it then factorises via SVD (with s_1 as the row index, (μ_1, α_1) as the column index)to obtain the unitary matrix U_1, the first tensor core of the new MPS. Multiplying $\mathbf{C}_1$ by $U_1^\dagger$ creates the tensor $\mathbf{R}_1$.



The procedure then carries on for sites $j = 2, \ldots, n-1$ just as before but carrying an extra index this time. Here, $\mathbf{C}_j$ is factorised by SVD with (β_j, s_j) as the row index, and (μ_j, α_j) as the column index.



When completed, tensors $U_1, U_2, \ldots, U_{n-1}, \mathbf{R}_n$ form the new MPS.

Certification of ownership of the copyright

Author	Conor O'Sullivan
Title	Efficient Simulation of Quantum Circuits using Tensor Networks: Approximating Grover's Algorithm as a Matrix Product Operator
Date of submission	17/04/24