UNIVERSITY OF CALIFORNIA
Los Angeles

# Efficient Quantum Block Encoding of Dense, Low Entropy Data

A thesis submitted in partial satisfaction
of the requirements for the degree
Master of Science in Computer Science

by

Jeffrey Chifang Ma

2024

ABSTRACT OF THE THESIS

Efficient Quantum Block Encoding of Dense, Low Entropy Data

by

Jeffrey Chifang Ma
Master of Science in Computer Science
University of California, Los Angeles, 2024
Professor Jens Palsberg, Chair

Efficient Quantum Block Encodings of non-unitary data are key for realizing quantum advantage for linear algebraic operations. Dense, low entropy matrices, which have a low amount of unique values with the distribution of values concentrated largely on one value, have a variety of applications such as DNA analysis. To enable efficient block encodings of dense, low entropy data, we introduce a new paradigm for developing quantum block encodings by reducing the block encoding problem to controlled quantum state preparation and quantum state preparation problems. Through this reduction, we are able to construct three novel block encoding algorithms that incorporate the elements of state preparation algorithms. We then evaluate these algorithms on dense, low entropy data and find that our three block encoding algorithms significantly reduce circuit depth with little ancilla and additional CNOT cost.

The thesis of Jeffrey Chifang Ma is approved.

Sriram Sankararaman

Jason Cong

Jens Palsberg, Committee Chair

University of California, Los Angeles

2024

# Table of Contents

**Preface** . . . . . **vii**

**1 Introduction** . . . . . **1**

1.1 Motivation . . . . . 1

1.1.1 QDNA Project . . . . . 1

1.1.2 QSVT . . . . . 2

1.2 Block Encoding . . . . . 3

1.3 UK Biobank Dataset . . . . . 3

1.3.1 Low Entropy Data . . . . . 4

1.4 Quantum State Preparation and Controlled Quantum State Preparation . . 5

1.5 Previous Work with Block Encodings . . . . . 6

1.5.1 Sparse Matrix Block Encodings . . . . . 6

1.5.2 Dense Matrix Block Encoding . . . . . 8

1.5.3 FABLE: Baseline Block Encoding Comparison . . . . . 10

**2 Block Encoding Based on Quantum State Preparation** . . . . . **12**

2.1 Equality between Quantum Data Loading Problems . . . . . 12

2.2 Reduction from State Preparation to Block Encoding . . . . . 14

2.3 Reduction from Block Encoding to Controlled QSP . . . . . 15

2.4 Reduction from Controlled QSP to QSP . . . . . 16

**3 Block Encoding Circuit Construction** . . . . . **22**

3.1 Column Based Block Encoding . . . . . 22

3.1.1 Linear Combination of Unitaries . . . . . 24

3.1.2 Frequency Based Centering for Low Entropy Data . . . . . 25
3.1.3 Example . . . . . 28
3.2 Top Down SP Inspired Block Encoding . . . . . 29
3.2.1 Alternating CQSP construction . . . . . 29
3.2.2 State Binary Tree and Angle Tree . . . . . 30
3.2.3 Top Down State Preparation . . . . . 31
3.2.4 Top Down Inspired Block Encoding . . . . . 32
3.2.5 Efficient Parallel Uniform Control Rotation Gate . . . . . 33
3.2.6 Circuit Depth . . . . . 35
3.2.7 Example . . . . . 35
3.3 BDD SP Inspired Block Encoding . . . . . 36
3.3.1 BDD Based State Preparation . . . . . 37
3.3.2 Converting the State Binary Tree to a Binary Decision Diagram For Block Encoding . . . . . 38
3.3.3 Efficient Path-based State Preparation for BDDs . . . . . 39
3.3.4 Frequency Based Centering . . . . . 39
3.3.5 Efficient Parallel MCNOT and MCRY . . . . . 40
3.3.6 Circuit Depth . . . . . 40
3.3.7 Example . . . . . 42

**4 Experimentation . . . . . 45**

4.1 Questions to answer . . . . . 45
4.2 Experimental Setup . . . . . 46
4.2.1 Input Data . . . . . 47
4.2.2 Baseline Algorithms . . . . . 47

4.3 Comparison of Block Encodings to Baseline Algorithms . . . . . 48
4.4 Scaling of Block Encodings with regards to input size . . . . . 50
4.5 Scaling of Block Encodings with regards to sparsity . . . . . 52
4.6 Overall evaluation . . . . . 54

**5 Conclusion . . . . . 56**

**A Appendix . . . . . 58**

A.1 Iten Multicontrol Scheme . . . . . 58
A.2 Linear Combination of Unitaries . . . . . 59
A.3 Uniform Rotation Gates . . . . . 59

**Bibliography . . . . . 60**

# Preface

I give my gratitude to my advisor, Jens Palsberg, for his guidance in writing this work and all the help over the years. I also thank the members of my committee for giving their time to read through and give feedback on this thesis.

To the UCLA Samueli School of Engineering, I am thankful for the Departmental Scholars program and for the opportunity to expand my breadth in the field of computer science.

Finally, I thank the friends and family who have helped me along the way. I thank my mom and dad for raising and caring for me all this time.

# CHAPTER 1

# Introduction

## 1.1 Motivation

Currently, one of the most promising areas in quantum computing is to significantly speed up linear algebraic calculations for higher-dimensional data. Quantum algorithms involving linear algebra, such as HHL [14], have demonstrated quadratic or even exponential speedup over their classical counterparts, leading to polylogarithmic time complexity with respect to the size of the input.

### 1.1.1 QDNA Project

DNA analysis is a critical field which benefits from said speedup. While the field is capable of analyzing large samples of individuals to find predictors of genetic diseases, such as prostate cancer [17], current DNA analysis algorithms cannot scale to meet the increasing size of biobanks as increasingly more genotype/phenotyope data of a population are added to the datasets. Quantum computing could bridge the gap between genetic differences in populations by allowing us to evaluate multiple massive datasets that span multiple populations.

As part of the qDNA project, we have identified three classical algorithms that can be enhanced via a quantum counterpart. These algorithms can identify enrichment of heritability of various disorders [1], infer population structure [25], and identify several genome-wide signals of recent positive natural selection [9] by performing data analysis on a SNP matrix. In each of the algorithms, we can replace key linear algebra steps with their quantum coun-

terparts. Specifically, by performing principal component analysis, trace estimation, and non-negative matrix factorization more efficiently via quantum computing, we can theoretically achieve an exponential speedup over the purely classical algorithms.

### 1.1.2 QSVT

The QSVT quantum algorithm is an optimal choice to implement the various required linear algebra mechanisms. QSVT (Quantum Singular Value Transformation) is based off the quantum signal processing framework introduced by Low et al [19]. Quantum signal processing enables arbitrary polynomial transformations on a scalar value by first encoding the scalar into the top of a unitary matrix $U(a) \in SU(2)$, then repeatedly applying both "signal-processing" single qubit rotations and the unitary. Gilyen et al. then demonstrated that quantum signal processing can be generalized to arbitrary matrices by encoding a matrix into a larger unitary matrix $U$ (known as a block encoding), then performing polynomial transformations on the singular values of said matrix via repeated applications of the signal processing operation and controlled applications of $U$ and $U^\dagger$ [12]. However, repeated applications of $U$ leads to leakage (due to having undefined behavior on the "garbage")— Low further refined this notion of a block encoding by applying "qubitization" to it, essentially creating a new block encoding that avoids this problem via a single ancilliary qubit [20].

The QSVT algorithm is both widely applicable and efficient, assuming a sparse block encoding. It has been demonstrated that many quantum algorithm staples, such as quantum search and amplitude amplification, can be unified back into a QSVT problem [22]. The core idea of QSVT is that we can create a non-unitary subsytem of a quantum system, and perform operations directly on said subsystem. Thus, there have been direct applications of QSVT to principal component analysis, non-negative matrix factorization and trace estimation [18], [11], [21]. All of these applications demonstrate quadratic or exponential speedup over their classical counterparts.

## 1.2 Block Encoding

A block encoding is the embedding of a (scaled) matrix into a sub-block of a larger unitary matrix. For our purposes, we assume that the matrix is embedded in the principal block of the larger matrix (ie the top left).

In a quantum circuit, this appears as follows:

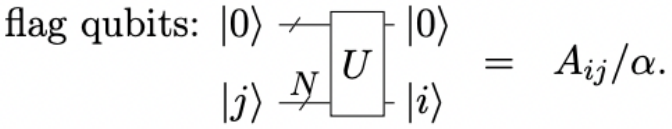


Figure 1.1: Example Block Encoding Circuit (Figure from Sunderhauf et al[28])

There are a variety of ways to block encode a matrix in a quantum circuit, as detailed in sections 3, 1.5.

## 1.3 UK Biobank Dataset

The efficiency of QSVT-based algorithms is predicated upon an efficient block-encoding representation of the data. For our purposes, we decided to focus on the UK Biobank dataset [5]. This dataset is rich with the genotype and phenotype data of 500,000 individuals, aged between 40 and 69. In the data matrix, each row represents a specific SNP (single-nucleotide polymorphism), and each column represents a specific individual. A data entry can take on the values of 0, 1, or 2, representing the count of mutations of a specific SNP inherited from the specific individuals mother or father. Specifically, 0 indicates no mutation for the SNP from the mother or father, 1 indicates a mutation from the mother or the father, and 2 indicates a mutation of the SNP from both the mother and the father. An example of the data is shown here 1.2:

We intentionally take a square matrix subset of the data for convenience in working with block encodings. The largest matrix we intend to work with is of size $2^{18} \times 2^{18}$, which

$$A = \underbrace{\begin{pmatrix} 2 & 0 & 1 & 2 \\ 1 & 0 & 2 & 2 \\ 1 & 2 & 2 & 1 \\ 2 & 2 & 2 & 2 \end{pmatrix}}_{\text{a column per individual}} \begin{matrix} \text{this row has data for SNP 1} \\ \text{this row has data for SNP 2} \\ \text{this row has data for SNP 3} \\ \text{this row has data for SNP 4} \end{matrix}$$

Figure 1.2: An Example 2x2 SNP matrix

corresponds to data on roughly a quarter million individuals and SNPs.

### 1.3.1 Low Entropy Data

An analysis of the data suggests that the SNP matrix has a lower entropy of data, specifically in terms of the columns. A table with the counts of each value is shown here 1.3:

| | Each row contains | | | Each column contains | | |
|---|---|---|---|---|---|---|
| | #0 | #1 | #2 | #0 | #1 | #2 |
| Minimum | 7 | 5,634 | 72,296 | 12,766 | 71,872 | 355,850 |
| 1st Quartile | 189 | 14,497 | 196,420 | 14,089 | 80,345 | 359,081 |
| Median | 912 | 30,785 | 259,571 | 14,219 | 80,645 | 359,342 |
| Mean | 9,121 | 51,708 | 230,444 | 14,223 | 80,632 | 359,351 |
| 3rd Quartile | 9,294 | 85,557 | 276,586 | 14,351 | 80,947 | 359,599 |
| Maximum | 73,262 | 147,164 | 285,612 | 16,511 | 84,164 | 368,963 |

Figure 1.3: Summary Statistics of UK Biobank Dataset from the QDNA Project

One characteristic of the data is that the distribution of 0's, 1's and 2's among the columns is fairly regular. This implies that most individuals carry roughly the same amount of mutations from their parents overall, irrespective of SNP. This characteristic of the data implies low entropy. The Shannon entropy of a data matrix can be defined as:

$H(A) = -\sum_{d \in A} p_d log(p_d)$

where $d$ corresponds to unique values of a matrix (in our case $d \in \{0, 1, 2\}$), and $p_d$ is the probability of seeing the value in the matrix. A lower entropy corresponds to a lower informational value of a message in information theory, and thus corresponds to a potentially more efficient way of representing the data. For example, we could solely represent the data by denoting the positions of just the 1s and 0s in each column– thus obtaining an $NMlogN$

representation of the data in a $N \times N$ SNP Matrix with sparsity $M$ (which denotes number of non-two values) in each column.

All in all, this data analysis suggests that dense, low entropy data might be a good candidate for block encodings that haven't been extensively studied in the past.

## 1.4 Quantum State Preparation and Controlled Quantum State Preparation

We now briefly discuss an alternate set of techniques to encode a classical dataset in the quantum realm. An amplitude encoding of a classical dataset consists of the following statevector:

$\psi = \sum_i^m \alpha_i |i\rangle$

where for some classical dataset $x^{(1)}, ..., x^{(m)}$ of $m$ data elements, $\alpha_i$ corresponds to the normalized $i$th element of $x$, and $|i\rangle$ correponds to the index of the $i$th element.

State Preparation involves preparing an arbitrary state on a set of qubits, similar to initializing a variable in classical computing. Typically, a state preparation algorithm constructs a unitary $U_{prep}$ such that:

$U_{prep}|0^n\rangle = |\psi\rangle$

where $|\psi\rangle$ is the target state. Similarly, controlled quantum state preparation involves a set of pre-initialized qubits $|i\rangle$, and a set of possible amplitude encodings $|\psi_0\rangle, ..., |\psi_k\rangle$. Controlled quantum state preparation generates a unitary such that:

$U_{cqsp}|i\rangle|0^n\rangle = |i\rangle|\psi_i\rangle . \forall i \in \{0,1\}^k$

These alternative quantum data loading problems are relevant due to a number of factors. For one, research into quantum state preparation is slightly more mature, simply because quantum block encodings are a newer concept. So, ideas from quantum state preparation can be applied to quantum block encoding. Furthermore, quantum state preparation can also give us a general idea of circuit depth for quantum data loading of the same data. Later on,

we also show that the problem of block encoding can also be reduced to controlled quantum state preparation and state preparation problems.

## 1.5 Previous Work with Block Encodings

Most previous works involving quantum block encodings have focused on sparse matrices and structured dense matrices. Furthermore, most previous works assume either oracle access or QRAM to load data values. For our goals, we want a clear idea of circuit depth, circuit width and subnormalization cost. Thus, we avoid using QRAM and oracle access in this work, but we consider such approaches here.

### 1.5.1 Sparse Matrix Block Encodings

In Gilyen et al's seminal paper [12], block encodings for certain sparse matrices were shown to be efficient, given oracle access. Further work in the area demonstrates that sparse block encodings can be represented using fewer qubits and clearer costs of oracle gates. Because all of the mentioned sparse block encodings use some form of oracle access, we decide to avoid comparing our block encoding constructions to these papers directly, and instead focus on the qubit and subnormalization cost of the introduced papers.

#### 1.5.1.1 Explicit Quantum Circuits for Block Encoding of Certain Sparse Matrices

Linlin's paper [6] introduces a general block encoding scheme for encoding sparse matrices through the use of two oracles, $O_A$ and $O_C$. The block encoding scheme is efficient in the number of qubits, only using s+1 qubits (where s is the log of the sparsity S), and has a subnormalization cost of $\alpha = S||A_{max}||$, where $A_{max}$ is the greatest value of the matrix. The paper also details several explicit constructions of specific types of matrices, such as banded circulant matrices and extended binary trees.

The $O_C$ oracle and the $O_A$ oracles act as sparse value row lookup per column and a sparse

value initialization for current sparse coordinate respectively. The core idea behind these oracles is that through a function $c(j, l)$ that gives the row index of the jth non-zero value in the lth column, one can avoid having to use more qubits to refer to a sparse value in a matrix. Specifically, the oracles take up the form:

$O_C|l\rangle|j\rangle = |l\rangle|c(j,l)\rangle$

$O_A|0\rangle|l\rangle|j\rangle = (A_{c(j,l),j}|0\rangle + \sqrt{1 - |A_{c(j,l),j}|^2}|1\rangle)|l\rangle|j\rangle$

By using a diffusion operator $H^{\otimes n}$ to search through all the possible sparse values, the paper is able to correctly initialize the amplitudes of the matrix through the $O_A$ oracle, then correctly initialize $|j\rangle$ to have the correct resulting row values through the $O_C$ oracle. A final diffusion operator uncomputes the ancilla qubits in order to properly move the matrix to the principal block.

While the explicit constructions of block encodings are not useful for our use case, the ideas of using Hadamard gates to search over the rows of the matrix and the use of an extra rotation qubit are important for our work and prevalent through the literature.

#### 1.5.1.2 Block-encoding structured matrices for data input in quantum computing

Sunderhauf et al. [28] construct sparse block encodings that are more focused on subnormalization. Similar to Linlin's paper, they reduce the number of required qubits by using a form of sparse indexes, this time by converting a sparse column index (corresponding to row of sparse value) and column to a data index and multiplicity "coordinate" and converting that into a spare row index (corresponding to column of sparse value) and row. Thus, the block encoding scheme uses three oracles: $O_r$ to get the sparse row index and row from a data index and multiplicity, $O_c^\dagger$ to get the data and multiplicty from a sparse column index and column, and $O_{data}$ to initialize the corresponding matrix value on a rotation qubit, just like with Linlin's paper. In this base scheme, it uses s+1 qubits and achieves a subnormalization of $\alpha = \sqrt{S_r}\sqrt{S_c}||A_{max}||$.

By removing the dependence of the $O_{data}$ matrix on $|j\rangle$, more quantum circuit machinery can be introduced. A delete oracle and delete qubit can remove the dependence of a certain amount of required numbers to delete out-of-range data and multiplicity "coordinates". A preamplification scheme (introduced by Gilyen et al [12]), can be achieved by separating the data oracle into two pieces to be singular value amplified. A PREP and UNPREP scheme can be used to commute out the $O_{data}$ oracle to reduce subnormalization again by moving the row/column oracles into a PREP and UNPREP oracle that act parallel with the diffusion operations, resulting in a smaller quotient in the oracle.

The quantum machinery introduced in this paper is in general useful for lowering the subnormalization of block encodings. We end up using a similar technique to the delete oracle in the BDD block encoding 3 through a path qubit that indicates whether a path is legitimate or not.

### 1.5.2 Dense Matrix Block Encoding

Most dense Block Encoding constructions also similarly use oracle access or QRAM, to construct the block encoding circuit. This is because in general block encodings for dense matrices are prohibitively expensive. Thus, either underlying structure of the matrix, oracle access, or QRAM must be used to reduce the required number of qubits and circuit depth.

#### 1.5.2.1 Quantum Resources Required to Block-Encode a Matrix of Classical Data

The Quantum Resources paper [10] considers the use of QRAM for encoding an arbitrary matrix, and details the implementation and resource estimation of such a QRAM block encoding. QRAM is specifically defined to perform the following action:

$\sum_j \alpha_j |0\rangle|j\rangle \xrightarrow{QRAM} \sum_j \alpha_j |j\rangle|b_j\rangle$

where $b_j$ is some classical data (like an index) loaded into a quantum state. The general strategy of the block encoding is to use two unitaries, $U_L$ and $U_R$, to load in a matrix's

normalized row and its row normalization relative to the overall Frobenius norm $||A||_F$. The overall subnormalization of this approach is the Frobenius norm. This general block encoding framework is directly related to our column block encoding approach described later 1, where we prepare a matrix's normalized columns instead.

The reduction of block encoding to controlled-quantum state preparation and thus state preparation is introduced in this paper but is not well explored (only considers the controlled state preparation of rows). We expand on this idea in the next chapter, which is the key behind our block encoding approach.

The Quantum Resources paper [10] introduces several QRAM variations (SS-QRAM, BB-QRAM) that have varying tradeoff of T-depth and T-count, as well as Fixed precision and Pre-rotated approaches to constructing the state preparation unitaries using the various previously-defined QRAMs. The paper focuses on T-count and T-depth as well as subnormalization. Overall, the paper gives an exact estimate of qubit count, t-depth and t-count for each QRAM and state preparation framework. In the event of QRAM being realized efficiently, this quantum resource paper probably provides the best depth/gate count for all reviewed dense block encodings.

#### 1.5.2.2 Block-encoding dense and full-rank kernels using hierarchical matrices: applications in quantum numerical linear algebra

Nguyen et al. [24] address block encodings for a certain subset of matrices called kernel matrices, which arise from discretizing or smoothing a kernel function $k(x, x')$. The paper produces block encodings that are efficient for matrices that are neither low-rank or sparse. It does this through hierarchical splitting (splitting the matrix along its off-diagonal blocks) and approximating each block with a low rank matrix. Through a kernel oracle $O_k|i\rangle|j\rangle|z\rangle = |i\rangle|j\rangle|z \oplus k(x_i, x_j)\rangle$, a kernel matrix K can be block encoded through hierarchical splitting into admissible blocks through two uses of the $O_k$ oracle and $polylog(\frac{N}{\epsilon})$ one and two-qubit gates. The admissible blocks, which are approximated with a low rank truncated Taylor series, are then combined through Linear Combination of Unitaries.

This approach can be generalized to arbitrary Hermitian matrices through an oracle $O_a|i\rangle|j\rangle|z\rangle = |i\rangle|j\rangle|z \oplus \alpha_{ij}\rangle$, where $\alpha_{ij}$ is a bit string description of $A_{ij}$. This uses two uses of the $O_a$ oracle, $polylog(\frac{N}{\epsilon})$ ancilla qubits and $polylog(\frac{N}{\epsilon})$ gates. This uses a generalized hierarchical splitting to split the matrix into sparse components.

In our approach to block encodings, we use the techniques of LCU and splitting of a state into sparse components inspired by this paper extensively.

### 1.5.3 FABLE: Baseline Block Encoding Comparison

The Fast Approximate BLock Encodings (FABLE) paper [8] is the only block encoding paper examined that proposes a generalized block encoding for any matrix without any oracle access or QRAM usage. Thus, it is the one we mainly refer to as the baseline for our block encoding constructions.

The FABLE construction is simple. It consists of a diffusion operator, a uniform controlled rotation gate on the rotation qubit, a swap network and a final diffusion operator. Thus, it has a qubit cost of (n+1) and a subnormalization cost of $2^n$. The block encoding matrix is defined as:

$U_A = (I_1 \otimes H^{\otimes n} \otimes I_n)(I_1 \otimes SWAP)O_A(I_1 \otimes H^{\otimes n} \otimes I_n)$

where $O_A|0\rangle|i\rangle|j\rangle = (A_{i,j}|0\rangle + \sqrt{1 - |A_{i,j)}|^2})|1\rangle|i\rangle|j\rangle$ is implemented with a uniform controlled rotation gate. The construction of this uniform controlled rotation gate is detailed in the Appendix A.3. A general overview of the construction is that it uses $O(N^2)$ alternating CNOTs and $Ry$ gates to encode a set of angles $\theta'$ which are related to the angles that encode the actual state $\theta$ by a linear system. This transforms $N^2$ multi control rotations for a dense matrix into a more manageable amount. Thus, the general idea of the FABLE block encoding is to search over all the possible rows, rotate to the correct values of the rows, swap in the resulting rows into the correct spot, and uncompute the ancilla.

FABLE also enables approximation by compressing the uniform controlled rotation gate. By removing some of the Ry gates with angles less than a cutoff threshold $\delta_c$, FABLE can

reduce the amount of Ry gates and cancel some of the CNOT gates at the cost of an error of $\epsilon = N^3\delta_c$. While for our purposes we mostly focus on implementing exact block encodings, we acknowledge that the topdown block encoding described in a later section can benefit from this tradeoff as well.

In our work, we use both the idea of the SWAP gate to properly initialize $|j\rangle$ as well as the idea of an efficient uniform controlled rotation gate. Furthermore, the reduction from block encoding to CQSP 1 directly reflects the FABLE block encoding construction, showing that FABLE itself is a reduction to controlled quantum state preparation.

# CHAPTER 2

# Block Encoding Based on Quantum State Preparation

We show that there are reductions between the various different quantum data loading problems of block encoding, state preparation and controlled state preparation. Specifically, the different quantum algorithms can be shown to be reduced to sub-problems of each other. By demonstrating this equality, we can more directly apply state preparation algorithms to block encoding problems. This is important as this avenue of thought is only briefly explored in a previous work [10], and it opens up the block encoding space to use algorithms from the slightly more mature field of quantum state preparation and apply them more directly.

## 2.1 Equality between Quantum Data Loading Problems

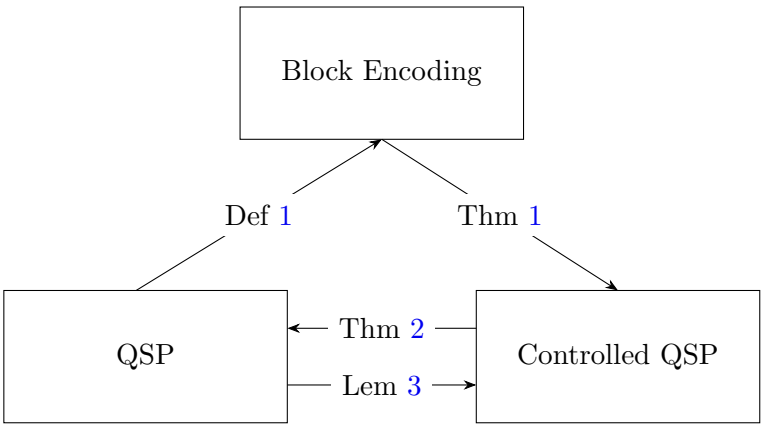


Figure 2.1: Reduction diagram between various quantum data input problems corresponding to the theorems, definitions and lemmas shown below

As shown above 2.1, we intend to show that a block encoding problem can be reduced to a controlled quantum state preparation problem, which in turn can again be reduced into

a quantum state preparation problem. In order to aid this discussion, we formally define the previously introduced data loading algorithms. We note that we only focus on encoding real-valued positive data.

**Definition 1** (Block Encoding)**.** *Let $a, n, m \in R$, $a = m + n$, $N = 2^n$. For a given $NxN$ matrix $A$ that operates on $n$ qubits, we construct a $(\alpha, m, \epsilon)$-block encoding of $A$ if*

$$||A - \alpha(\langle 0^m| \otimes I_N)U_A(|0^m\rangle \otimes I_N)||_2 \leq \epsilon$$

Note that there are three additional variables of a block encoding that are relevant to the overall circuit complexity, in addition to the circuit depth and cnot cost. First, the sub-normalization of the matrix $\alpha$ directly leads to an increase in the number of shots as $\alpha$ indicates the probability of measuring a garbage result (non-zero ancilla qubits). A larger sub-normalization decreases the probability of reading a relevant result at all. Second, $m$, which denotes the number of ancilla qubits, directly corresponds to circuit width and thus qubit cost. Finally, if $\epsilon > 0$, then the block encoding is an approximate block encoding and additional shots must also be used to account for the introduced error.

While various quantum state preparation algorithms introduce similar costs as well, the current literature does not define it as formally for state preparation as for block encodings. We introduce alternate definitions for state preparation and controlled state preparation to account for this.

**Definition 2** (Quantum State Preparation (SP))**.** *For a given amplitude encoding $|\psi\rangle$ that occupies $n$ qubits, $U_{sp}$ is a $(\alpha, m, \epsilon)$-state preparation of $|\psi\rangle$ if*

$$|||\psi\rangle - \alpha(\langle 0^m| \otimes I_N)U_{sp}(|0^m\rangle \otimes I_N)|0^n\rangle||_2 \leq \epsilon$$

*where $|0^n\rangle$ is simply equivalent to $|e_1\rangle$*

Note that state preparation differs from block encodings in that the sub-normalization is often required to be the norm of the vector itself, i.e. the necessary sub-normalization to encode the data in the quantum realm. This means that state preparation introduces a sub-normalization of 1 compared to the normalized vector. In contrast, block encodings do

not need to have a sub-normalization related to the normalized vector, and thus have a more relaxed sub-normalization requirement. Instead, block encodings must strictly uncompute the garbage of the ancilla qubits such that they measure 0, so the matrix is encoded in the top-left of the unitary. For state preparation, the garbage of the ancilla qubits doesn't matter.

For our purposes, we relax the requirement of sub-normalization of state preparation, such that state preparation can introduce an arbitrary amount of sub-normalization to encode an un-normalized vector. We also enforce that a state preparation algorithm doesn't introduce garbage in ancilliary qubits, so it fits the block encoding scheme better.

Finally, controlled quantum state preparation depends on both a given set of amplitude encodings as well as the initialization of additional qubits denoted $|i\rangle$

**Definition 3** (Controlled Quantum State Preparation (CQSP))**.** *For a given set of amplitude encodings $|\psi_0\rangle, ..., |\psi_k\rangle$, $U_{csp}$ is a $(\alpha_i, m, \epsilon)$-controlled state preparation of $|\psi_0\rangle, ..., |\psi_k\rangle$ if*

$\forall i \in \{0,1\}^k . ||| i\rangle|\psi\rangle - \alpha_i(\langle 0^m| \otimes I_N) U_{csp} (|0^m\rangle \otimes I_N)|i\rangle|0^n\rangle||_2 \leq \epsilon$

*where the sub-normalization $\alpha_i$ depends on $|i\rangle$. $m$ denotes additional qubits beyond the initial $n$ needed to encode the data.*

One can think of the controlled quantum state preparation problem as encoding the set of amplitudes in a staggered fashion across the unitary matrix (such that they correspond to the different i's).

Now, with a more consistent set of definitions, we can formally prove the reduction between the different quantum data loading algorithms. We assume that we are working with exact algorithms for now.

## 2.2 Reduction from State Preparation to Block Encoding

It's simple to reduce a quantum state preparation problem to a block encoding problem. For a target state $|\psi\rangle$, one can simply construct a random matrix A such that the leftmost

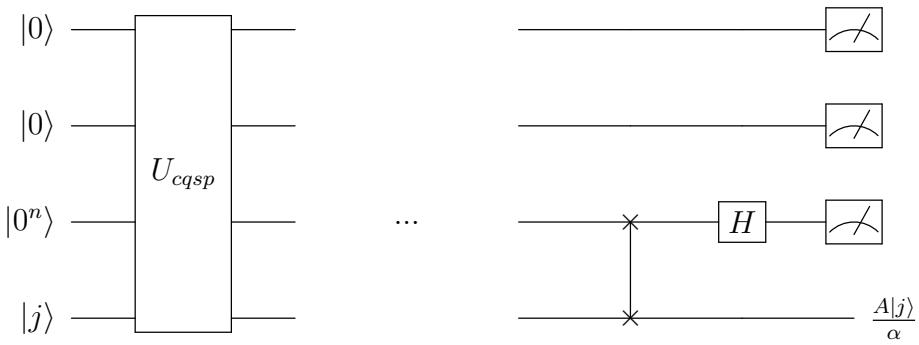


Figure 2.2: Circuit construction of Block Encoding using CQSP unitary

column in the matrix is equal to $|\psi\rangle$. Thus, one can construct a state preparation circuit of $|\psi\rangle$ with one use of a block encoding unitary, with some additional greater subnormalization cost. This is simply a result of our $(\alpha, m, \epsilon)$ definitions 1, 2.

## 2.3 Reduction from Block Encoding to Controlled QSP

**Theorem 1** (Reduction from BE to CQSP)**.** *Given an algorithm $CQSP(|\psi_0\rangle, ..., |\psi_k\rangle)$ that finds a $(\alpha_i, m)$-controlled state preparation of $|\psi_0\rangle, ..., |\psi_k\rangle$ and encodes it via unitary $U_{csp}$, one can construct a $(\alpha_1, m+1, \epsilon)$-block encoding of a $NxN$ matrix $A$ called $U_A$ as follows:*

$U_A = (I_1 \otimes H^{\otimes n})(I_1 \otimes SWAP)(U_{csp})$

*where the constructed $U_{csp}$ is outputted from $CQSP(\beta_1 A|e_1\rangle, \beta_2 A|e_2\rangle, ...\beta_n A|e_n\rangle)$ for input $j$ and SWAP indicates a swap network between qubits $|j\rangle$ and $|0^n\rangle$ ancillas. $\beta_i$ indicates the required normalization such that the values in one vector are equivalent to the values in the vector with the max sub-normalization. Specifically, $\beta_i = \alpha_i/\alpha_{max}$. The construction is shown on a quantum circuit in Figure 2.2.*

*The reduction introduces one ancilla qubit for easier normalization, and introduces an additional normalization of $\sqrt{2}^n * \alpha_{max}/\alpha_i$. Thus, the resulting block encoding is a $(\sqrt{2}^n * \alpha_{max}, m+n+1)$-block encoding of $A$.*

*Proof.* Assume wlog that m=0 as by our assumptions any additional qubits must be uncomputed to 0 anyway.

We need to verify that

$\langle 0|\langle 0|^{\otimes n}\langle i|U_A|0\rangle|0\rangle^{\otimes n}|j\rangle = \frac{1}{\sqrt{2}^n * \alpha_{max}} A_{ij}$

On the right part of the inner product we have

$|0\rangle|0\rangle^{\otimes n}|j\rangle$

$\xrightarrow{U_{csp}} \frac{\alpha_j}{\alpha_{max}}|0\rangle A|e_j\rangle|j\rangle + \sqrt{1 - \frac{\alpha_j}{\alpha_{max}}^2}|1\rangle A|e_j\rangle|j\rangle$

$\rightarrow \frac{\alpha_j}{\alpha_{max}}|0\rangle \sum_{k=0}^{N-1} \frac{A_{kj}}{\alpha j}|k\rangle|j\rangle + \sqrt{1 - \frac{\alpha_j}{\alpha_{max}}^2}|1\rangle \sum_{k=0}^{N-1} \frac{A_{kj}}{\alpha j}|k\rangle|j\rangle$

Considering only the "correct" case where ancillas are 0, we get

$\rightarrow \sum_{k=0}^{N-1} \frac{A_{kj}}{\alpha_{max}}|0\rangle|k\rangle|j\rangle$

$\xrightarrow{SWAP} \sum_{k=0}^{N-1} \frac{A_{kj}}{\alpha_{max}}|0\rangle|j\rangle|k\rangle$

On the left part of the inner product we have

$|0\rangle|0\rangle^{\otimes n}|i\rangle$

$\xrightarrow{H^{\otimes n}} \frac{1}{\sqrt{2^n}} \sum_{s=0}^{N-1} |0\rangle|s\rangle|i\rangle$

The inner product then yields

$\langle 0|\langle 0|^{\otimes n}\langle i|U_A|0\rangle|0\rangle^{\otimes n}|j\rangle$

$= \frac{1}{\sqrt{2^n}}(\sum_{s=0}^{N-1}\langle 0|\langle s|\langle i|)(\sum_{k=0}^{N-1} \frac{A_{kj}}{\alpha_{max}}|0\rangle|j\rangle|k\rangle + \sum_{k=0}^{N-1} \sqrt{1 - \frac{\alpha_j}{\alpha_{max}}^2}\frac{A_{kj}}{\alpha_{max}}|1\rangle|j\rangle|k\rangle)$

$= \frac{1}{\sqrt{2}^n * \alpha_{max}} \sum_{s=0}^{N-1}\sum_{k=0}^{N-1} A_{kj}\langle s||j\rangle\langle i||k\rangle$

$= \frac{1}{\sqrt{2}^n * \alpha_{max}} A_{ij}$ □

Thus, with one use of a CQSP unitary, one can construct a block encoding. This also shows that the FABLE block encoding construction is performing a CQSP construction through its use of the Uniform Controlled Rotation Gate.

## 2.4 Reduction from Controlled QSP to QSP

**Theorem 2** (Reduction from CQSP to SP)**.** *Given an algorithm $QSP(|\psi\rangle)$ that finds a $(\alpha, m)$ state preparation of $|\psi\rangle$ and encodes it as unitary $U_{sp}$, one can construct a $(\alpha_1, m)$-controlled*

*quantum state preparation circuit $U_{csp}$ for a set of states using only applications of the SP unitaries or controlled SP unitaries.*

*Proof.* We show by induction that a CQSP circuit can be constructed via QSP unitaries.

Note that CQSP is defined as performing the following operation:

$|i\rangle|0^n\rangle \rightarrow |i\rangle|\psi_i\rangle$

For the base case of $i \in 0$, there is only state $|\psi_0\rangle$ that needs to be prepared. Thus, the CQSP problem directly translates into a QSP problem, and we can directly apply one use of $U_{sp_0}$ to get the desired result. This is also known as the canonical CQSP problem.

Assume that we can construct a CQSP circuit from SP unitaries for $|i\rangle \in \{0,1\}^n$. Then, we can construct a CQSP circuit for $|j\rangle \in \{0,1\}^{n+1}$ by constructing two CQSP subproblems. First, we split the Hilbert space into two parts based on the value of the additional qubit of j.

$|j\rangle|0^n\rangle = \alpha|0\rangle|i\rangle|0^n\rangle + \beta|1\rangle|i\rangle|0^n\rangle$

Then, we can apply two controlled CQSP unitaries that are constructed off of $|\psi_0, ..., \psi_i\rangle$ and $|\psi_{i+1}, ..., \psi_k\rangle$ with no additional SP cost. Note that this scheme assumings the same subnormalization between the amplitude encodings of the CQSP unitaries (ie we can choose $\alpha$ to be the same for the two CQSP unitaries).

$(|0\rangle\langle 0| \otimes U_{cqsp_0} + |1\rangle\langle 1| \otimes U_{cqsp_1})|j\rangle|0^n\rangle = \alpha|0\rangle|i\rangle|\psi_{i0}\rangle + \beta|1\rangle|i\rangle|\psi_{i1}\rangle$

□

This somewhat trivial theorem has some important implications. First, this theorem is the basis for the column-based block encoding construction we detail in a later section 5, enabling an easy way to generate many different block encoding algorithms by plugging in different kinds of state preparation algorithms. This is shown by the following corollary 2.1.

**Corollary 2.1** (Column-indexed CQSP)**.** *For a given set of amplitude encodings $|\psi_0\rangle, ..., |\psi_k\rangle$, one can construct a Controlled QSP given a QSP algorithm as follows:*

$U_{csp_i} = \prod_{i=0}^{k}(|i\rangle\langle i| \otimes U_{sp_i})$

*where $U_{sp_i}$ is the resulting unitary of $QSP(|\psi_i\rangle)$*

Second, this theorem can be extended by considering the problem of controlled quantum state preparation as a subset of quantum state preparation problems. One key idea of this is that controlled quantum state preparation can be thought of as a "partial" state preparation problem.

$|0^n\rangle|0^n\rangle \xrightarrow{SP} |j\rangle|0^n\rangle \xrightarrow{partial} |j\rangle|\psi_j\rangle$

**Lemma 3.** *Given an algorithm $CQSP(|\psi_0\rangle, ..., |\psi_k\rangle)$ that finds a $(\alpha_i, m)$-controlled state preparation of $|\psi_0\rangle, ..., |\psi_k\rangle$ and encodes it as unitary $U_{csp}$ as well as a $(\alpha, m)$ state preparation unitary $U_i$ that prepares state $|i\rangle$, one can construct a $(\alpha, m)$-state-preparation of an amplitude encoding $|\psi\rangle$ as long as $|\psi\rangle$ can be decomposed in the following manner:*

$|\psi\rangle = \sum_{i\in\{0,1\}^k} |i\rangle|\psi_i\rangle$

$U_{sp} = (U_{csp})(U_i \otimes I^{\otimes n})$

*In other words, if we can separate out the initialization of the "i" qubits and the target state $|\psi\rangle$ spans all possible values of i in some fashion, then the problem can be reduced into a simpler SP and CQSP problem.*

Another way of thinking about this is that the initialization of $|i\rangle$ can be commuted out and thus does not need to be initialized later, only needed to be used for entangling the state. This also suggests that controlled state preparation spans an "easier" problem space than general state preparation for the same number of qubits.

With these findings, we can construct a CQSP circuit by alternating between CQSP and QSP problems. For this purpose, we consider the subset of exact QSP algorithms that are constructed a set of qubits at a time sequentially. These "head recursive" or "tail recursive" algorithms largely fall into two categories: those that construct the amplitude encoding by inputting one state at a time, and those that construct the encoding by decomposing the state and constructing the decomposed states. We formally define such algorithms like so:

**Definition 4** (Recursive State Preparation)**.** *A state preparation algorithm is a recursive state preparation if, for a given amplitude encoding $|\psi\rangle$ that occupies n qubits, it constructs*

*the $(\alpha, m, \epsilon) - U_{sp}$ circuit by calling itself as a subroutine in some manner. The number of subproblems of a state preparation algorithm depends on how many times the Hilbert space of $|\psi\rangle$ is split up on each iteration. Specifically, a state preparation algorithm that decomposes the state can be classified as recursive if the algorithm forming the resulting state preparation unitary consists of the form:*

$U_{sp} = U_{sp_n}$

$U_{sp_i} = ENTANGLE((I^{\otimes_{ic}} \otimes U_{p_i} \otimes I^{\otimes_{(n-i)c}}), \{U_{sp_{i-1,0}}, ..., U_{sp_{i-1,j}}\})$

*where $U_{p_i}$ denotes a state preparation unitary for c qubits and the ENTANGLE algorithm produces a circuit that entangles the orthonormal subspace of $U_{p_i}|0\rangle$ with the states of $U_{sp_{i-1,0}}|0\rangle, ..., U_{sp_{i-1,j}}|0\rangle$ such that the product of the resulting constant factors equal to the intended constant factors of $|\psi\rangle$.*

*So the resulting state of applying the unitary to $|0\rangle$ is of the form:*

$U_{p_i}|0^c\rangle = |\psi_i\rangle = \sum_l^{2^c-1} \alpha_{il}|l\rangle$

$U_{sp_i} = \sum_l^{2^c-1} \alpha_{il}|l\rangle U_{sp_{i-1,j}}|0^{(n-i)c}\rangle = \sum_l^{2^c-1} \sum_j \alpha_{il}\beta_{(i-1)lj}|l\rangle|j\rangle = |\psi\rangle = \sum_r \gamma_r|r\rangle$

*Note that $U_{sp_0}$ denotes a base case that is unique to the specific algorithm, which may add additional circuitry.*

*State preparation algorithms which construct the overall state one basis state at a time can be constructed similarly, just at the state-level.*

$U_{sp} = \prod_s U_{state_s}$

$U_{state_s} = U_{sp_{sn}}$

$U_{sp_{si}} = ENTANGLE((I^{\otimes_{ic}} \otimes U_{p_{si}} \otimes I^{\otimes_{(n-i)c}}), \{U_{sp_{si-1,0}}, ..., U_{sp_{si-1,j}}\})$

We can show that for recursive state preparation problems that operate on a set of qubits at a time, a "control" qubit of a cqsp built on the recursive state preparation can be loaded similarly to a "target" qubit of the qsp problem.

**Theorem 4** (Alternate CQSP construction)**.** *Given a recursive algorithm $QSP(|\psi'\rangle)$ that constructs a $(\alpha, m)$ state preparation of $|\psi'\rangle = \sum_r \gamma_r|r\rangle$ by calling itself as a subroutine,*

*one can can modify the QSP construction to construct a CQSP on the control qubit in the following manner:*

$$U_{sp_i} = ENTANGLE(|0\rangle\langle 0| \otimes I^{\otimes n} \otimes Ry_0 + ... + |c\rangle\langle c| \otimes I^{\otimes n} \otimes Ry_c, \{U_{sp_{si-1,0}}, ..., U_{sp_{si-1,j}}\})$$

*where* $Ry_0, ..., Ry_c$ *introduce rotations on a separate ancilla qubit in order to adjust the subnormalization of the state to be equal. Specifically,* $Ry_l|0\rangle = \alpha_j|0\rangle + \sqrt{1-\alpha_j^2}|1\rangle$ *where* $\alpha_j = norm_j/norm_{max}$ *corresponding to the norms of* $U_{sp_{si-1,j}}|0\rangle$.

*Proof.* First, we refer to Lemma 3 and note that a CQSP problem can be represented as a QSP problem with the prior initialization of the control qubit. Thus, we just need to show that

$$ENTANGLE(|0\rangle\langle 0| \otimes I^{\otimes n} \otimes Ry_0 + ... + |c\rangle\langle c| \otimes I^{\otimes n} \otimes Ry_c, \{U_{sp_{si-1,0}}, ..., U_{sp_{si-1,j}}\})(I^{\otimes ic} \otimes U_i \otimes I^{\otimes (n-i)c})$$

$$= ENTANGLE((I^{\otimes ic} \otimes U_{p_i} \otimes I^{\otimes (n-i)c}), \{U_{sp_{i-1,0}}, ..., U_{sp_{i-1,j}}\})$$

where $\exists U_{p_i}$ for all possible $U_i$ such that the above equality is true up to some degree of normalization (as we can account for that in the subnormalization cost).

Note that $U_i$ is an arbitrary matrix allowing for any possible states within the Hilbert space of c qubits while $U_{p_i}$ encodes information relevant to the state itself, specifically $U_{p_i}|0\rangle = \sum_l^c \alpha_l|l\rangle$, where $\alpha_l = \frac{norm_l}{\sqrt{\sum_p^c norm_p^2}}$.

For the LHS we have:

$$ENTANGLE(|0\rangle\langle 0| \otimes I^{\otimes n} \otimes Ry_0 + ... + |c\rangle\langle c| \otimes I^{\otimes n} \otimes Ry_c, \{U_{sp_{si-1,0}}, ..., U_{sp_{si-1,j}}\})(I^{\otimes ic} \otimes U_i \otimes I^{\otimes (n-i)c})$$

$$= (\sum_l^c |c\rangle\langle c| \otimes Ry_c \otimes U_{sp_{i-1,c}})(I^{\otimes ic} \otimes U_i \otimes I^{\otimes (n-i)c})$$

$$\xrightarrow{\text{Applied to } |0\rangle}$$

$$(\sum_l^c |c\rangle\langle c| \otimes Ry_c \otimes U_{sp_{i-1,c}})(I^{\otimes ic} \otimes U_i \otimes I^{\otimes (n-i)c})|0^c\rangle|0^n\rangle$$

$$= (\sum_l^c |c\rangle\langle c| \otimes Ry_c \otimes U_{sp_{i-1,c}})\frac{1}{norm_l}\sum_l^c \alpha_l|l\rangle|0^n\rangle$$

$$= \frac{1}{norm_l norm_{max}}\sum_l^c \sum_j \alpha_l\beta_{jl}|l\rangle|j\rangle$$

By setting $U_{p_i} = U_i$, on the RHS we have:

$$ENTANGLE((I^{\otimes ic} \otimes U_{p_i} \otimes I^{\otimes (n-i)c}), \{U_{sp_{i-1,0}}, ..., U_{sp_{i-1,j}}\})$$

$$\xrightarrow{\text{Applied to } |0\rangle}$$

$$= \frac{1}{\sqrt{\sum_p^c norm_p^2}} \sum_l \sum_j \alpha_l \beta_{jl} |l\rangle |j\rangle$$

□

Thus we have shown the ability to reduce a QSP problem to a CQSP problem and vice versa, at least for a subset of QSP problems. This enables a different way of thinking about block encoding problems, making more "inline" ways of preparing cqsp unitaries which ultimately make block encoding unitaries.

# CHAPTER 3

# Block Encoding Circuit Construction

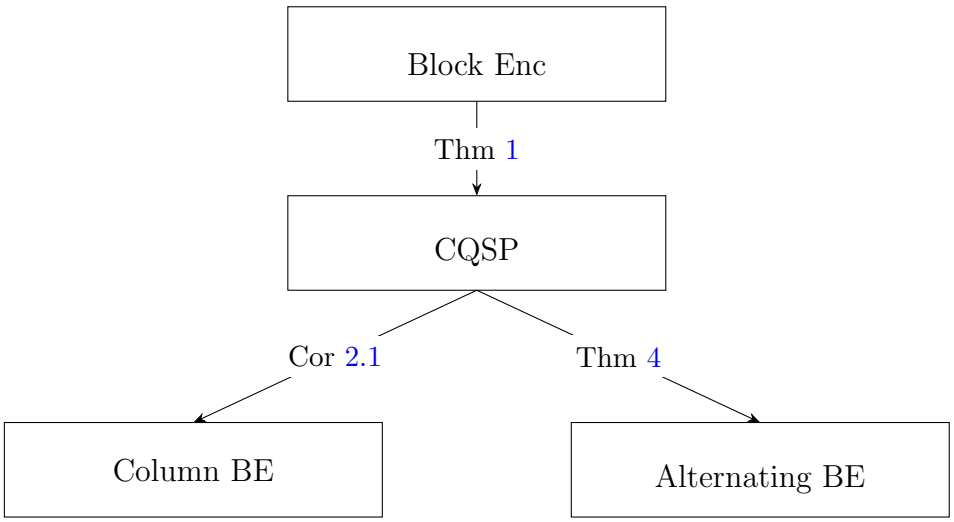


Figure 3.1: Block Encoding Construction Process showing the various reductions taken to construct the BE

Corollary 2.1, Theorem 1 and Theorem 4 form the basis for our new block encoding constructions 3.1. First, we reduce the block encoding construction to a QSP problem using Theorem 1. Then, we can apply either the the column based block encoding construction 2.1 or we can recursively construct the CQSP from a given state preparation algorithm 4.

## 3.1 Column Based Block Encoding

By the prior Corollary 2.1, we can formally construct a "column-based" block encoding scheme. This idea of splitting the matrix into separate state preparation vectors is similar to the idea proposed in [10], but is more formally supported by the proofs of the prior

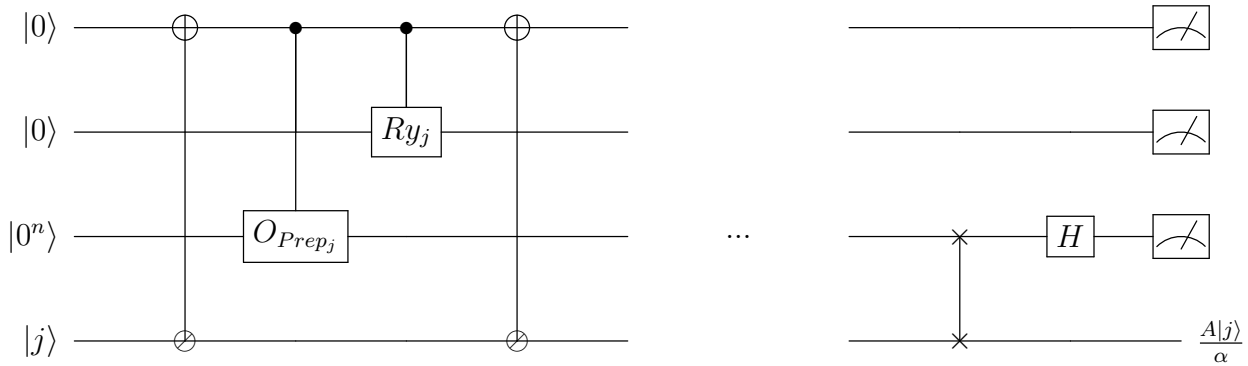

Figure 3.2: Column Block Encoding Circuit, where j denotes a column of the matrix

chapter. The main motivation behind column-based block encoding for our purposes is the regularity in the data distribution over the columns (individuals) of the SNP data set. If we can guarantee a certain amount of entropy in the columns, which we later show can be translated to sparsity of the matrix, then we could theoretically get a better circuit depth.

We formally define the column-based block encoding algorithm like so:

**Definition 5** (Column Block Encoding)**.** *Given a $N \times N$ matrix $A$ that operates on $n$ qubits and an QSP algorithm $QSP(|\psi\rangle)$ that generates an $(\alpha, c, \epsilon)$-state preparation circuit given an arbirary $|\psi\rangle$, we construct a $(\sqrt{N} * \alpha_{max}, l+n+c+2, \epsilon$ column-based block encoding by combining Corollary 2.1 and Theorem 1. $l$ ancilla qubits are used for the different modifications we introduce below.*

$$U_{be} = \prod_j^N MCX(q_0..q_n, t_q)(|1\rangle\langle 1| \otimes Ry_j \otimes O_{prep_j} \otimes I^{\otimes n})MCX(q_0..q_n, t_q)$$

*For each column of the matrix $A_j$, we generate a state preparation circuit $O_{prep_j}$ that constructs the amplitude encoding of the column. Depending on the value of $j$, the state preparation is selectively applied via a multi-control CNOT gate denoted by the symbol $\oslash$, with correct anticontrols and controls corresponding to the bit representation of $|j\rangle$. We apply a controlled state preparation circuit by using the correctly toggled multi-control ancilla qubit. This successfully entangles each resulting column vector $|\psi_j\rangle$ with its corresponding column index $|j\rangle$, and thus results in a correct CQSP of $(A|e_1\rangle, ..., A|e_N\rangle)$. The required $\beta_j$ sub-normalization is introduced by a controlled Ry gate. Thus, by Theorem 1 we have a block encoding of $A$.*

With this simple but general definition, we can take advantage of the large suite of existing quantum state preparations and generate different variations of block encodings that scale differently according to data distribution. Furthermore, this also gives us a baseline circuit scaling of $O(N^2 + NlogN)$, with $N$ columns generating $2N$ multicontrol CNOTs (which have a depth linear in the number of qubits) and with N quantum state preparation circuits with an upper bound of $\frac{23}{24}N$ circuit depth complexity [31].

However, we find in practice that the scaling of the circuit depth complexity is higher than the baseline fable implementation. This is because the uniform controlled rotation gate is highly efficient whereas the multi-control CNOT and the controlled application of the state preparation circuit can introduce a dominating amount of constant factors. Thus it is preferable to pursue sparse state preparation problems, where circuit depth can be significantly lower relative to a large input size.

### 3.1.1 Linear Combination of Unitaries

One way to enforce a sparser state is to split the state based on the value of the column entry. This way, while the circuit depth scales more with the number of unique values in the column, the sparsity for each state preparation algorithm is higher. We employ the linear combination of unitaries technique from block encoding literature to enforce sparsity, described more in Appendix A.2. This is done by using $d = log(D)$ additional ancilla qubits, where $D$ denotes the number of unique data values in a column. We employ a state preparation pair in the following form:

$$P_L|0\rangle = \sum_i^D \frac{\sqrt{count_i} * v_i}{\sqrt{\sum_j^D count_j * v_i^2}}|i\rangle$$
$$P_R|0\rangle = \frac{1}{\sqrt{D}} \sum_i^D |i\rangle$$

where $count_j$ denotes the number of values in the column with the value corresponding to index $j$. With this state preparation pair, we can employ linear combination of unitaries with state preparations of the "binary" states $|\psi_0\rangle, ..., |\psi_D\rangle$ where the states are binary representations of the indexes of the value in the original matrix. Thus, while $|\psi_0\rangle, ..., |\psi_D\rangle$

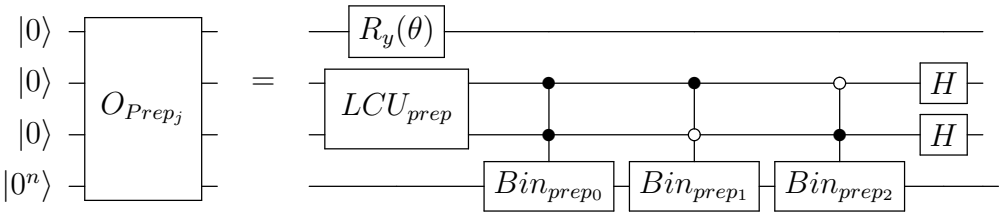


Figure 3.3: Linear Combination of Unitaries Circuit for encoding SNP column via binary states

should span the whole subspace, the individual states themselves are relatively sparser.

We note that the linear combination of unitaries introduces a greater sub-normalization cost– however the sub-normalization matters less depending on the dominance of the most frequent value and the relative size of the most frequent value. That is, the state preparation algorithm now produces a $(\sqrt{\sum_j^D count_j * v_i^2} * \sqrt{2}^d, c+d, \epsilon)$ state preparation circuit 3.3 with the added modifications.

Note that in this situation we can drop the binary state preparation circuit corresponding to a 0 value, as the empty basis states don't need to be initialized.

By using an additional D qubits instead of d qubits, we can reduce the added depth from the controlled applications of unitaries by using only one control instead of d. We call this "wide" binary state preparation for now. Furthermore, by adding some "padding" to the LCU input such that the result of $P_L|0\rangle$ does not include the basis state $|0\rangle$, we can avoid having to control the application of the binary state preparations when we construct the controlled $O_{Prep_j}$ unitary. Instead, we can apply controlled-$LCU_{prep}$ unitaries then apply the controlled binary state preparation unitaries as in normal Linear Combination of Unitaries circuits.

### 3.1.2 Frequency Based Centering for Low Entropy Data

We can take this concept a step further by assuming a "dominance" of the most frequent data value, like in the case of low entropy data. This means that all the other data values have very sparse binary states. In this case, we can instead center our data around the most

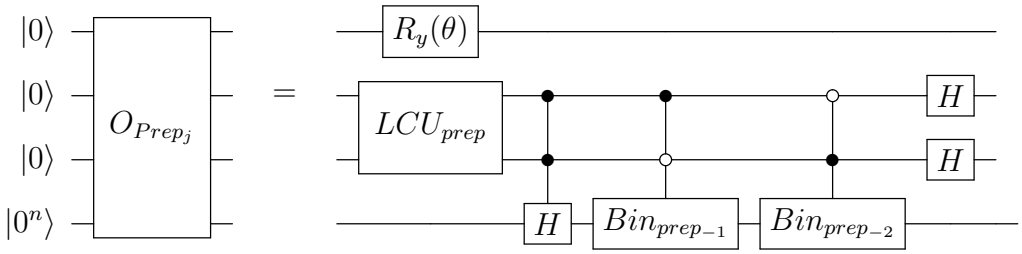


Figure 3.4: Linear Combination of Unitaries Circuit with frequency centering for encoding SNP column via sparse binary states

frequent value. Specifically, we subtract all the other values by the most frequent value. This makes the state preparation for the most frequent case trivial, as we simply prepare the state by spanning the whole subspace of n qubits with a simple diffusion operation (Hadamard gates). The addition between the binary state of the most frequent element and the other values subtracted by the most frequent element corresponds to the original values. Thus, we only have to deal constructing efficient state preparation circuits for the resulting sparse binary states 3.4.

Just like before, the frequency based centering results in an additional subnormalization cost. We end up having to use a similar state preparation pair. Let r denote the index of the most common value, and $D' = D \backslash \{r\}$

$$P_L = \frac{\sqrt{N} * v_r}{\sqrt{N * v_r^2 + \sum_j^{D'} count_j * (v_j - v_r)^2}} |r\rangle + \sum_{i \in D'} \frac{\sqrt{count_i} * (v_i - v_r)}{\sqrt{N * v_r^2 + \sum_j^{D'} count_j * (v_i - v_r)^2}} |i\rangle$$

$$P_R = \frac{1}{\sqrt{D}} \sum_i^D |i\rangle$$

Thus we now have a $(\sqrt{N * (v_i - v_r)^2 + \sum_j^{D'} count_j * v_i^2} * \sqrt{2}^d, c + d)$ state preparation circuit. The block encoding itself introduces an additonal $\sqrt{N}$ subnormalization and $n$ qubits.

**Algorithm 1** Column Block Encoding Algorithm

**Input:** QSP algorithm that returns $(\alpha, c)$-sp for data size $N$, input data matrix $A \in \mathbb{R}^N \times \mathbb{R}^N$

**Output:** Unitary $U_{be}$ which is $(\alpha, n+c+l+2)$-block encoding of A

$n \leftarrow log(N)$,

$d \leftarrow MaxCountUniqueValues(A), D \leftarrow 2^d$

$$l \leftarrow \begin{cases} d & \text{if lcu prep} \\ D & \text{if wide bin prep} \\ 0 & \text{else} \end{cases}$$

$q \leftarrow QuantumRegister(2 * n + c + l + 2 \text{ qubits})$

$maxNorm \leftarrow max(||Ae_j||)$

$t_q \leftarrow q_{2*n+c+l+1}, r_q \leftarrow q_{2*n+c+l+1}$ ▷ Denotes target qubit and rotation qubit

**for** j in 1..N **do** ▷ Prepare the state of each column

$mcx(q1..q_{n-1}, t_q)$

$|1\rangle\langle 1|(t_q) \otimes P_L(q_{2*n+c+1}..q_{2*n+c+l})$

$$states \leftarrow \begin{cases} BinStates(A|e_j\rangle) & \text{if lcu prep} \\ BinStates(A|e_j\rangle - MostFreqElement(A|e_j\rangle)) + H & \text{if freq centering} \\ A|e_j\rangle & \text{else} \end{cases}$$

**for** i, state in 1..len(states), states **do**

$|\phi\rangle \leftarrow StatePrep(state)$

$|i\rangle\langle i|(q_{2*n+c+1}..q_{2*n+c+l}) \otimes QSP(|\phi\rangle, q_n..q_{2n+c}) \otimes Ry(2cos^{-1}(|||\phi\rangle||/maxNorm))(r_q)$

**end for**

$|1\rangle\langle 1|(t_q) \otimes P_R(q_{2*n+c+1}..q_{2*n+c+l})$

$mcx(q1..q_{n-1}, t_q)$

**end for**

**for** i in 1..n **do**

$SWAP(q_i, q_{n+i})$

$H(q_{n+i})$

**end for**

### 3.1.3 Example

$$A = \begin{bmatrix} 2 & 0 & 1 & 2 \\ 0 & 0 & 2 & 2 \\ 1 & 2 & 2 & 1 \\ 2 & 2 & 2 & 2 \end{bmatrix}$$

For our examples, we walk through the process of encoding (a portion of) of the above $4 \times 4$ example SNP matrix. We demonstrate column block encoding by encoding the first column of the example matrix using the linear combination of unitaries and frequency based centering techniques. This results in the following circuit 3.5.

First, we check the input $|j\rangle$ and conditionally mark a flag qubit $|0_{flag}\rangle$ with a multi control CNOT gate, specifically a toffoli with two anticontrols (for the first column) in our case. Then we examine the column to prepare and construct the LCU state preparation pair. For the first column $2|00\rangle + 0|01\rangle + 1|10\rangle + 2|11\rangle$ with unique values $v_1 = 0, v_2 = 1, v_3 = 2$, the most frequent element is $v_3 = 2$ (note that the value indexes are shifted to avoid the 0 index). Thus, we center around the value of 2 and only have to construct state preparation unitaries to prepare the binary states of $|10\rangle$ and $|01\rangle$ corresponding to the indexes of the values $v_1 = -2, v_1 = -1$ respectfully. These binary states can be simply prepared with a not gate on the first and second qubit respectively.

We must also construct a unitary to encode the LCU left unitary $P_L|00\rangle = \frac{\sqrt{4}*2}{\sqrt{21}}|11\rangle - \frac{\sqrt{1}*1}{\sqrt{21}}|10\rangle - \frac{\sqrt{1}*2}{\sqrt{21}}|01\rangle$. In the example, this is done by constructing a controlled top-down state preparation consisting of $R_y$ gates. The LCU unitary is conditionally applied based off the flag qubit. Then, the binary state preparation unitaries are conditionally applied based off the lcu qubits. Because the 0 index is avoided, we know that if the lcu qubits are not $|0\rangle$ then the flag qubit is set, and can thus avoid an additional control. Finally, the $P_R$ unitary, which corresponds to a diffusion operation across the lcu qubits, is conditionally applied based off the flag qubit as well, and the flag qubit is uncomputed in order to allow for setting up the next column.

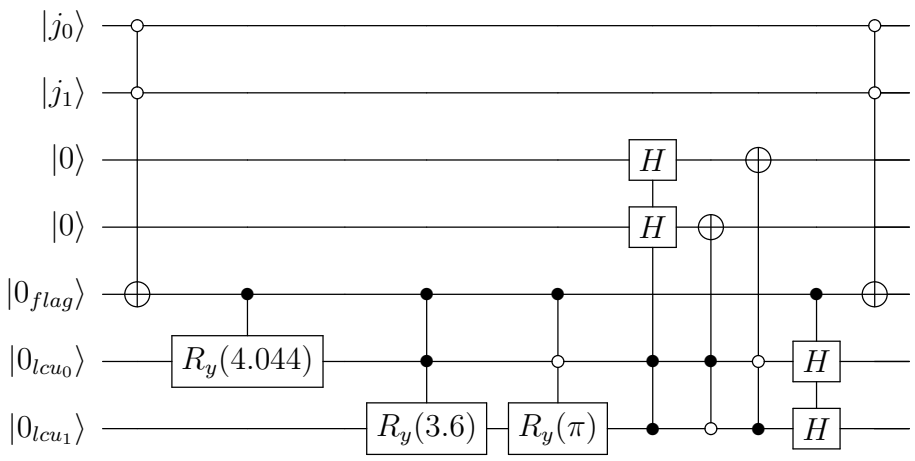


Figure 3.5: CQSP portion of the top-down block encoding circuit for the example matrix

## 3.2 Top Down SP Inspired Block Encoding

Theorem 4 suggests that the column-based block encoding is simply a subset of the possible CQSP algorithms that can be generated by the reduction of cqsp to qsp and vice versa. We demonstrate the strength of this revelation by showing an alternating CQSP construction based off the top down state preparation algorithm, which simply prepares a state based off a state binary tree which ultimately determines the resulting angles of a uniform controlled rotation gate.

### 3.2.1 Alternating CQSP construction

From Theorem 4, we know that recursive state preparation algorithms can allow for a more "inline" CQSP construction. We further demonstrate that we can construct a state preparation circuit in an alternating fashion as described below:

**Corollary 4.1.** *Given a recursive algorithm $QSP(|\psi'\rangle)$ that constructs a $(\alpha, m)$ state preparation of $|\psi'\rangle$ by calling itself as a subroutine, one can construct a CQSP of states $|\psi_0\rangle, ..., |\psi_k\rangle$ and a given $|j\rangle$ where $j \in \{0,1\}^k$ in the following manner:*

$$U_{sp} = U_{sp_n}$$

$$U_{sp_i} = \begin{cases} ENTANGLE(|0\rangle\langle 0| \otimes I^{\otimes n} \otimes Ry_0 + |1\rangle\langle 1| \otimes I^{\otimes n} \otimes Ry_1, \{U_{sp_{si-1,0}}, U_{sp_{si-1,1}}\}) \\ \quad \textit{if i refers to a "control" qubit} \\ ENTANGLE((I^{\otimes ic} \otimes U_{p_i} \otimes I^{\otimes (n-i)c}), \{U_{sp_{i-1,0}}, ..., U_{sp_{i-1,j}}\}) \\ \quad \textit{otherwise} \end{cases}$$

*where if i is even, then it corresponds to a control qubit of* $|j\rangle$

We now look at top-down state preparation [4] as a potential candidate recursive state preparation algorithm to use for this alternating approach.

### 3.2.2 State Binary Tree and Angle Tree

For a state preparation problem that prepares a state $|\psi\rangle = \sum_p \alpha_p |p\rangle$, some existing methods of state preparation can be understood as walks down a state binary tree which represents the encoded data, with top-down state preparation as one such example. Essentially, each branch of the binary tree represents a split of the Hilbert space into two subspaces, where each have a different overall "magnitude". After n layers of the tree, we are left with $2^n$ subspaces corresponding to the different bases of the n qubit subspace (the indexes of the values), with the magnitude of the leaf nodes being equal to the corresponding initial value of the state (ie $\alpha_p$ for index p). An example of a state binary tree and angle tree is shown here 3.6:

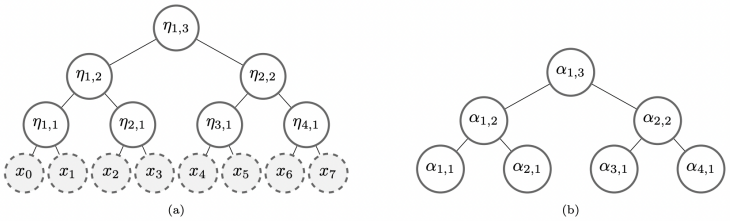


Figure 3.6: Example state tree and angle tree (Figure from paper [4])

The non-leaf nodes can be thought of as subproblems of its parent node (with the root

node being the entire state preparation problem) corresponding to the state preparation states entangled with $|1\rangle$ and $|0\rangle$ respectively, where the "left" child on the state tree denotes the state entangled with $|1\rangle$. The corresponding magnitude of each of these non-leaf nodes is determined by the magnitudes of its own subproblems. Specifically, the magnitude of a node is determined like so:

$$\nu_{i,k} = \sqrt{\nu_{i+1,k} + \nu_{i+1,k+1}^2}$$

The state binary tree can then be converted to an angle tree which contains the rotation angles for Ry gates needed to allocate probability to each respective subproblem. This is determined by performing a simple calculation.

$$\theta_{i,k} = 2 * cos^{-1}(\nu_{i+1,k}/\nu_{i,k})$$

That is, we simply determine the angle based off how much we allocate probability to the "right" case which corresponds to a $|0\rangle$ state of the qubit.

By conjecture, we suggest that tree-based state preparation algorithms are inherently recursive.

**Proposition 1.** *Let $QSP(stateTree)$ be a state preparation algorithm that constructs a state from some form or derivation of a state binary tree. Then, QSP can be defined in the form of Definition 4, where the children of a node denote the $U_{sp_{i-1,0}}$ and $U_{sp_{i-1,1}}$ unitaries and the state is prepared one qubit at a time.*

### 3.2.3 Top Down State Preparation

Top down state preparation is a linear transformation that performs a sequence of uniformly controlled rotations based off the angles of each layer of the angle tree. Because it performs these rotations sequentially in a level-order traversal of the angle tree, it is called a top-down approach. First, the initial state is initialized to match the rotation of the root node. Then, to load the states into the next level, the current state is sequentially combined with the values of the next state. Thus, the state is loaded one qubit at a time in a recursive fashion 3.7, matching the requirements of the prior lemma.

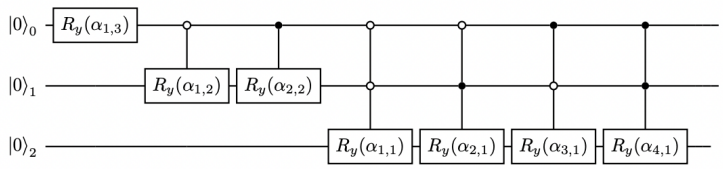


Figure 3.7: Top Down State Preparation for 3 qubits (Figure from paper [4])

This generates n Uniform Controlled rotations with increasing number of controls up to n, thus generating a circuit depth of $O(\sum_i^N i)$. Thus, this is a $(norm, 0)$ exact sp algorithm.

### 3.2.4 Top Down Inspired Block Encoding

We can construct a encoding scheme based off the prior lemma for recursive state preparation algorithms.

On the state binary tree, we differentiate between nodes for control and target qubits. Control qubits are already "prepared" for us, so we don't have to rotate the control qubits themselves. Thus, they don't have an "angle" for allocating probability, and we can cut out the uniform controlled rotations for the control qubits. Thus, with the alternating scheme, we can "cut out" half the rotation gates corresponding to the control qubits. In the alternating scheme as proposed, this corresponds to the even qubits.

We pay for the cost of not preparing the control qubits ourselves by having an additional subnormalization cost, as detailed in the earlier theorem. At ctrl nodes, we still have to selectively allocate more or less probability mass due to the irregularity of the data (ie difference in normalization between two amplitude encodings). Thus, for the state preparation tree, we adjust the magnitude to correspond to the max of the subproblem magnitudes instead of the square root of the sum of the magnitudes squared for a control node.

$\nu_{i,k} = \max(\nu_{i+1,k}, \nu_{i+1,k+1}), \forall i$ st i is ctrl

This way, we also don't have to introduce a greater subnormalization cost as compared to a state preparation problem of the same size. However, this magnitude does not correspond to an angle of the angleTree node, but instead to an added variable of the node that we dub

node "subnormalization" or norm. A subnormalization at a corresponding node (given that the parent is a control node) is based on the subnormalization of the parent node as well as the ratio between the magnitude of the parent and the magnitude of the current node. Otherwise the subnormalization is the same as the parent node's.

$norm_{i,k} = norm_{i-1,k} * (\nu_{i-1,k}/\nu_{i,k})$

which leads to a corresponding angle on the rotation qubit.

We note that this rotation is only necessary at leaf (terminal) nodes as the angle tree's subnormalization already accounts for the normalization of the parent nodes in the variable itself.

We define the full algorithm in pseudocode 2.

While this block encoding scheme does ultimately result in the same kind of scaling as FABLE of $O(N^2)$, we note that we are effectively able to almost halve the circuit depth with a minimal increase in the number of CNOTs (from performing the parallel uniform control rotation gate). The reason why we are able to gain this advantage is because the state preparation algorithm splits the state into two per target qubit. Thus, splitting the last target qubit only uses a unifrom control rotation gate of n-1 controls rather than n controls for FABLE.

### 3.2.5 Efficient Parallel Uniform Control Rotation Gate

While there is an additional subnormalization cost for the control qubits, we show that the normalization depth cost can be mostly amortized into the last uniform controlled rotation. This can be shown in the following circuit 3.8:

Because the Uniform RY Gate consists of $2^n$ CNOTs targeted on a single target qubit, we can easily "parallelize" another uniform RY gate on the rotation gate. We simply just convert each CNOT of the original uniform RY gate into two CNOTs, one on the rotation qubit and one on the target qubit. Thus, by using the Walsh-Hadamard transform and gray code permutation ordering as shown by FABLE A.3, we can effectively parallelize the

**Algorithm 2** Top Down Block Encoding Algorithm

**Input:** input data matrix $A \in \mathbb{R}^N \times \mathbb{R}^N$

**Output:** Unitary $U_{be}$ which is $(\alpha, n+1)$-block encoding of A

$reorderedA \leftarrow MatrixOrder(A)$

$q \leftarrow QuantumRegister(2 * n + 1 \text{ qubits})$

$qubitOrder \leftarrow q_n, q_{2n}, q_{n-1}, q_{2n-1}..$

$stateTree \leftarrow StateDecomposition(reorderedA, qubitOrder)$

$angleTree \leftarrow AngleTree(stateTree, qubitOrder)$

$TopDown(angleTree, \{\})$

**procedure** TopDown(node, controlNodes)

**if** Node does not exist **then**

exit

**end if**

$targetNodes \leftarrow children(targetNodes)$ ▷ node if targetNodes is empty

**if** node does not refer to a control qubit **then**

**if** node is leaf **then**

$UCRY(targetNodes.normAngles, controlNodes.qubits, node.qubit)$

**end if**

$UCRY(targetNodes.magAngles, controlNodes.qubits, node.qubit)$

**end if**

$controlNodes \leftarrow controlNodes + node$

TopDown(node.left, controlNodes)

**end procedure**

**for** i in 1..n **do**

$SWAP(q_i, q_{n+i})$

$H(q_{n+i})$

**end for**

operation, as the two CNOTs operate on two different qubits.

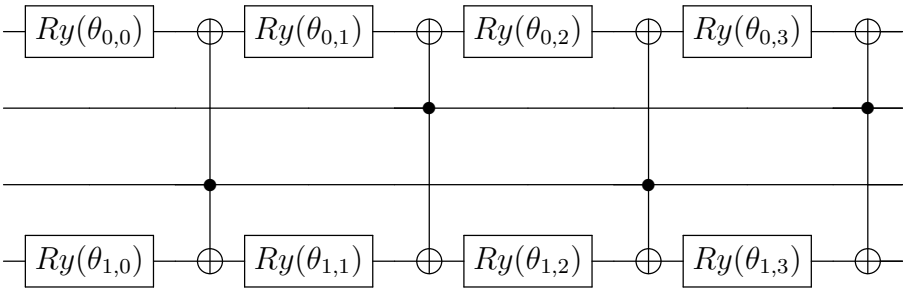


Figure 3.8: Parallel Uniform RY Gate on two qubits

### 3.2.6 Circuit Depth

The circuit consists of $n$ uniform control rotation gates with $1, 3, ..., 2n-1$ control qubits per gate. The last uniform control gate requires two rotations; however, as discussed above, this can be done efficiently in the same circuit depth. Thus, we have an overall circuit depth of $2 + 8 + ... + 2^{2n-1} = \frac{2^{2n}-2}{3}$. at most, which can be reduced by overlapping more CNOTs.

### 3.2.7 Example

We walk through an example for the CQSP portion of the top down block encoding circuit of the same SNP matrix A of the prior example. First, we construct the state tree 3.2.7 and angle tree 3.2.7 of the matrix. Note that because we construct in an alternating manner between control and target qubits, we must reorder the input matrix in order to match the qubit ordering. This is why the order of the leaf nodes of the state tree are different from

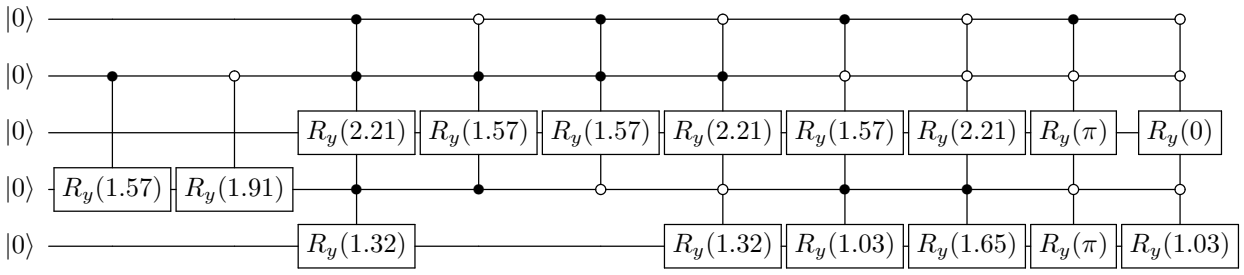


Figure 3.9: CQSP portion of the top-down block encoding circuit for the example matrix

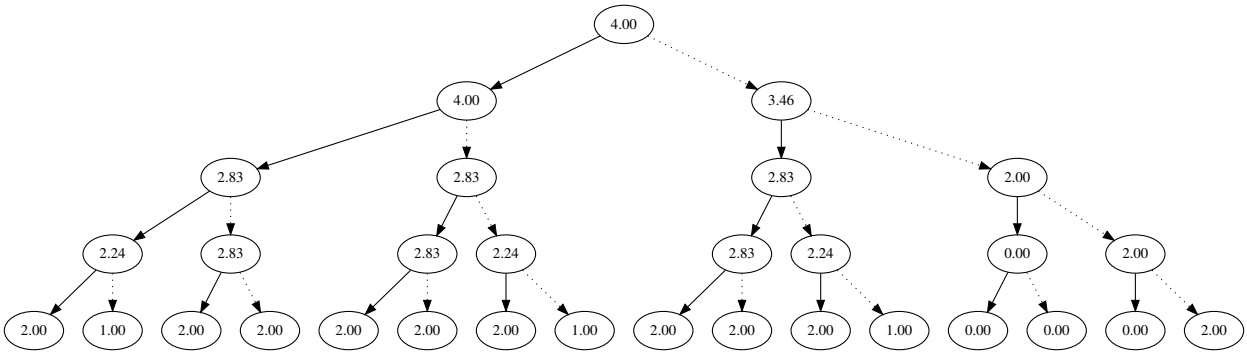


Figure 3.10: State Binary Tree representing the example SNP matrix

the order of simply the unraveled matrix. Every odd level of the state tree corresponds to a row index (target qubit), and every even level of the state tree corresponds to a column index (control qubit) in decreasing order.

After constructing the angle tree, the resulting circuit 3.9 is constructed in a linear fashion. For the "target" qubit levels, an Ry gate corresponding to each node is controlled by the qubits on the path to the node in the tree, with 0 branches corresponding to anticontrols and 1 branches corresponding to controls. The "control" qubit levels are skipped. At the last level, an additional Ry gate is needed for each node to introduce the overall sub-normalization for each path. Note that this Ry gate is also responsible for preparing empty values because the state preparation cannot zero out values (must be populated by at least one value).

## 3.3 BDD SP Inspired Block Encoding

While the top down inspired block encoding scheme seems to be the current best option for dense unstructured block encodings, beating out the scaling of the current SOTA FABLE, the circuit depth still scales quadratically with the size of the matrix. Instead, we can choose a sparse state preparation algorithm and convert them into a block encoding scheme.

We choose the BDD state preparation algorithm [23] as it naturally lends itself to the state preparation tree and angle tree construction described earlier.

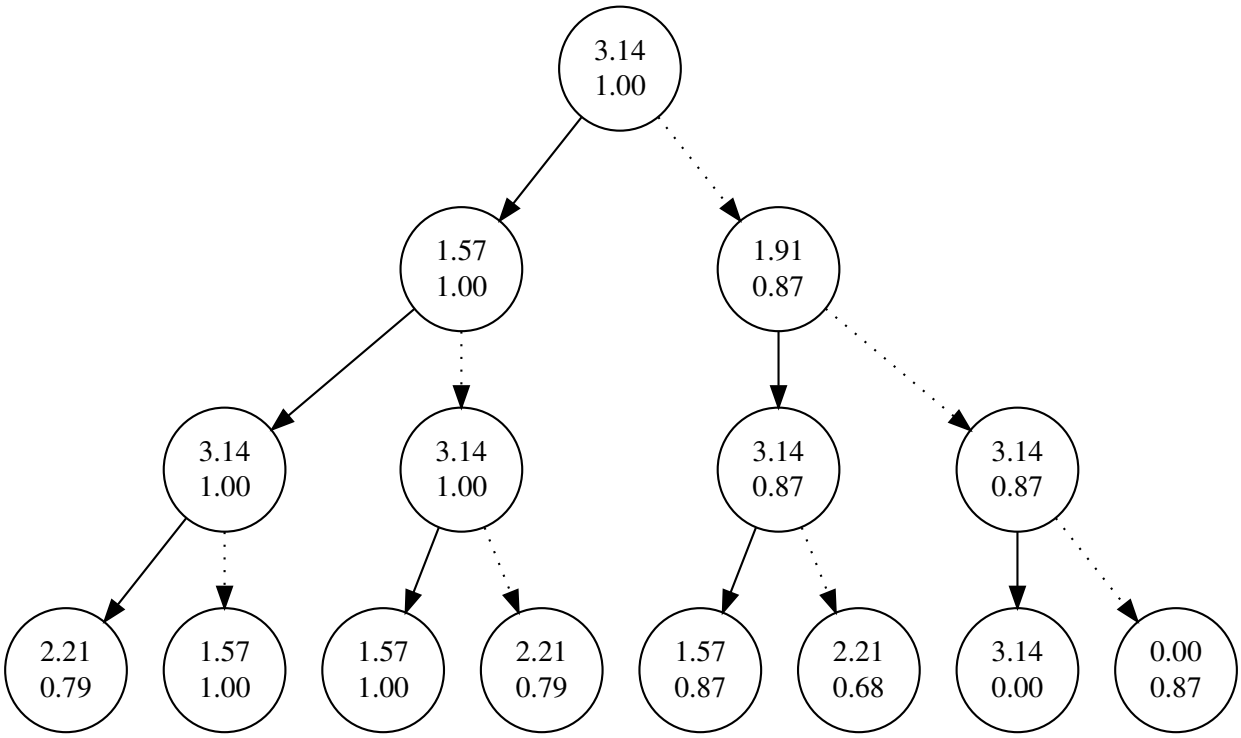


Figure 3.11: Angle Tree derived from the State Binary Tree

### 3.3.1 BDD Based State Preparation

BDD state preparation builds a state preparation given a BDD, more specifically a Reduced Ordered Binary Decision Diagram, of the state which we derive from the state binary tree. BDD state preparation is constructed in a path-wise fashion in a pre-order traversal. Gates are appended to the circuit for each path, with an ancilla "path" qubit determining whether the path had been prepared or not. This allows the paths to be prepared without interfering with existing paths.

In order to construct the state preparation, first the path qubit is initialized to $|1\rangle$ to indicate that the current path is not yet computed. Then, a path is traversed in a pre-order fashion, taking the 1 branch first. For the current node q in the traversal, we do as follows. First, we check if there are any reduced nodes between q and its parent. For each reduced node, a 2-controlled Ry gate with angle $\pi/2$ on the reduced node qubit is applied with

controls corresponding to the path qubit and the last node on the path that has a one-child taken. Then, we initialize q depending on q's node characteristics. If q is a branching node, which means it has both a zero-child and a one-child (next possible values of qubit can be 0 or 1), we apply to the quantum circuit, a 2-controlled Ry gate on q with the path qubit and the last node on the path that has a one-child as control qubits, and the angle being determined by the angle tree. Otherwise, q either has a one-child or a zero-child. If q has a one-child, then we apply a Toffoli on q with the same control qubits as before. Otherwise, we do nothing.

In order to finish initializing the path, we must also "commit" it by applying a multi-control CNOT to the path qubit. This marks the path as computed, and prevents other path computations from interfering with the existing state. The controls of the multicontrol CNOT correspond to the branching nodes on the path, with controls or anticontrols corresponding to if the one or zero branch was taken.

### 3.3.2 Converting the State Binary Tree to a Binary Decision Diagram For Block Encoding

Converting the state binary tree to a state BDD is a simple task. A BDD can be obtained from the state binary tree via a reduction process. In reverse level order, we can either merge two nodes in the same level or delete nodes. Two nodes are merged (with incoming edges redirected to the merged nodes) if they are terminal nodes with the same magnitude or if they are non-terminal nodes with the same sub-graphs (ie same subproblems). A node can be deleted if its left and right child point to the same child node.

The angle BDD is constructed from the BDD in the same manner as the angle tree, with some additional considerations for the case of preparing it for a block encoding. If nodes are deleted between a target node and its child, then the calculated angle should be based on the magnitude of the highest deleted node. For subnormalization calculation, iterate through all the levels from the level above the child's level to the parent level and keep track of what the magnitude should be. If the level corresponds to a control qubit, multiply its

subnormalization to the current subnormalization.

### 3.3.3 Efficient Path-based State Preparation for BDDs

We modify the original state preparation algorithm in a similar way to the topdown encoding. After constructing the angle BDD as described above, with the added modifications to have correct subnormalization and angle calculation, we construct the block encoding by following the general outline of the BDD state preparation algorithm. When we encounter a control node on the path, we treat it like a branching node but do not initialize it. Thus, control qubits contribute to the commit multicontrol and the various multicontrol rotations for initializing the state, the gates do not operate on the qubit themselves. One thing to note is that now instead of only considering the last 1 branch node, we have to consider a list of them, only "consuming" the list on initializing a target qubit in some way. This makes it so that we can account for 1 childs of control nodes, which don't necessarily indicate that the current state matches the state of the qubit. Thus, instead of a 2-ctrl Ry or Toffoli like in BDD state prep, we utilize either a multi control Ry or MCX.

### 3.3.4 Frequency Based Centering

We can use the same LCU technique in order to force the matrix to be sparser. We do this at the matrix-level granularity instead of the column-level granularity for efficiency, and we can choose to only eliminate the most common case. The state preparation pairs are of the following form:

$$P_L = \frac{\sqrt{N} * v_r}{\sqrt{sparseMag^2 + N * v_r^2}}|0\rangle - \frac{sparseMag}{\sqrt{sparseMag^2 + N * v_r^2}}|1\rangle$$
$$P_R = \frac{1}{\sqrt{2}} \sum_{i=0}^{1} |i\rangle$$

where sparseMag corresponds to the root node of the reduced and recalculated BDD after centering. The most common element is deleted from the BDD by simply removing its corresponding node and removing the edges that connect to it. The BDD is recalculated in a post-order fashion, with the magnitude of the subproblems being calculated before the

magnitude of the current node. This matches the prior construction, with the magnitude being adjusted for the deleted nodes through a power of two.

### 3.3.5 Efficient Parallel MCNOT and MCRY

We can modify the Iten construction A.1 to efficiently parallelize the construction as well. Note that a controlled Ry gate can be constructed like so 3.12:

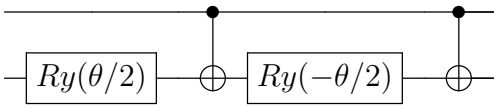


Figure 3.12: Controlled RY rotation gate Decomposition

In the Iten construction for CNOTs with $n \geq 5$, ancilla dirty qubits $n/2$, we have an "action" part and a "reset" part of the circuit, where the action toggles corresponding qubits allowing the target qubit to be correctly set, while the reset part untoggles the qubits. Note that toggling and untoggling the qubits just corresponds to doing two CNOTs. Thus, we can use the prior construction to first do a ry gate on the rotation qubit, do the "action" part with parallel toffolis on the target and rotation qubit, then do a ry gate and reset the rotation qubit as well.

### 3.3.6 Circuit Depth

The BDD consists of $k \leq NM$ paths. Thus, the resulting circuit consists of a sequence of up-to-n-controlled gates and a parallel multicontrol CNOT and controlled rotation gate. We note that for the sequence of up-to-n controlled gates, even in the worst case each controlled gate operates on a separate set of controls, and thus we only need O(n) CNOTs. Computing the ancilla qubit and rotation qubit is done with the above construction in O(n) CNOTs as well. Thus, the overall circuit depth is $O(klogN)$.

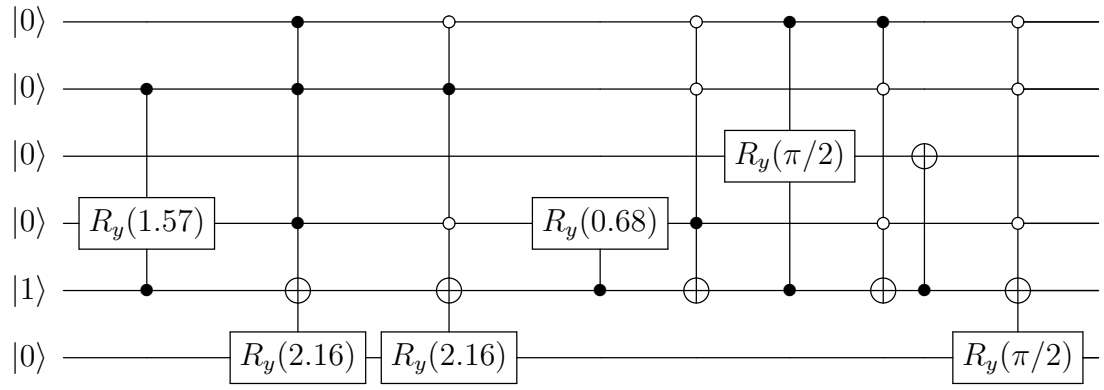


Figure 3.13: CQSP portion of the top-down block encoding circuit for the example matrix

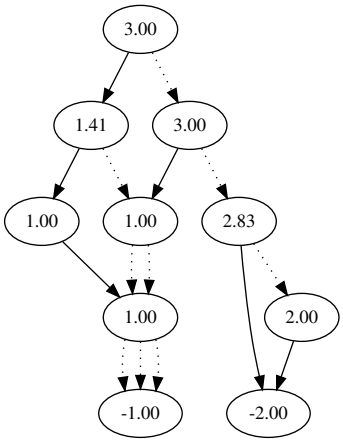


Figure 3.14: Derived BDD for the example SNP matrix

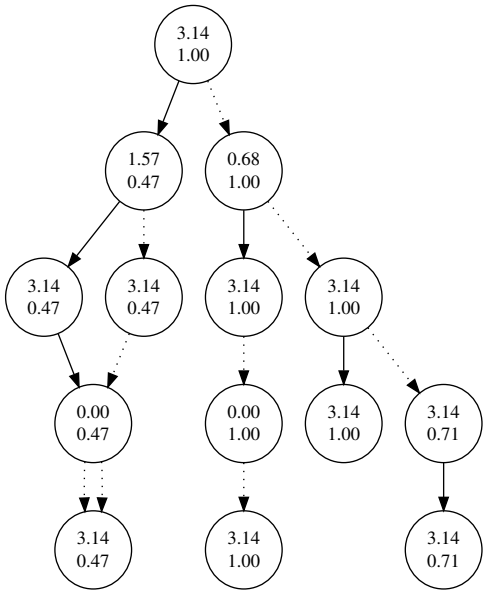


Figure 3.15: Angle Tree paths derived from the BDD

### 3.3.7 Example

We once again construct the CQSP portion of the circuit for the example SNP matrix. We can construct the following BDD 3.14 from the state preparation tree shown previously 3.2.7. First, we can force the BDD to encode a sparse state via frequency centering. This means that we remove the leaf nodes of the most common magnitude, two, and merge the resulting leaf nodes with the new difference magnitudes. This can be done either before or after the BDD is constructed. We construct the BDD bottom up, following reduction rules and magnitude calculation rules when applicable. For example, on the first level above the leaf node, we merge the first, fifth and sixth nodes from the left in the level into one node as they all point to the same child of 0 on the "one" path and "-1" on the "zero" path. We can also delete the seventh node in the same layer, as its two children point to the same "-2" value node. For the second level above the leaf node, we can merge the second and third nodes from the left as they have the same subtrees.

Once the BDD is constructed, we can construct the angle tree 3.15 in the same fashion as in the topdown state preparation construction, via post-order traversal. One thing to note

is that while two paths might traverse along the same nodes, their subnormalizations might differ. This can be seen with how the middle path on the angle tree does not merge back into the left nodes like how it does in the BDD.

We then are able to construct the CQSP portion of the block encoding 3.13 by following the paths of the angle tree. This results in 5 multi control CNOTS corresponding to each path with four other controlled gates that prepare either a branch or a one-child of a path. The encoding of the first 2 paths works like so: First, we start at the root control node and add it to the list of last1Nodes, as we take the one-child left path first. Then, we encounter a branching target node, and thus add a multi-controlled Ry gate with angle corresponding to the node's angle in the angle tree, with controls being the path qubit and the last1Node qubits. For the left path from this node, we clear its last1Node list and add only the target qubit that we just initialized, as we know that the target can only be initialized on the current path. From there, we encounter no other branching or one-child target nodes, and thus just add the commit parallel multicontrol CNOT and ry gates. On both the paths, we have incurred the same subnormalization of 0.47, which corresponds to a rotation on the rotation qubit of $\theta = 2 * cos^{-1}(0.47) = 2.16$. Note that while the control nodes in the second layer from the root node are non-branching, we still have to account for them in the commit multicontrol, as the control qubit initialization is not dependent on the current path.

**Algorithm 3** BDD Block Encoding Algorithm

**Input:** input data matrix $A \in \mathbb{R}^N \times \mathbb{R}^N$

**Output:** Unitary $U_{be}$ which is $(\alpha, n+1)$-block encoding of A

Prepare stateTree and qubitOrder the same as in TopDown algorithm

Get Reduced BDD and angleTree using same initial stateTree

BDD(angleTree, {}, {})

**procedure** BDD(node, currentPath, last1Nodes)

**for** Each reduced node between last node and node that is not control **do**

$MCRY(\pi/2, p_q + lastOneNodes.qubits, node.qubit)$

**end for**

$last1NodesL \leftarrow last1Nodes$

**if** node does not refer to a control qubit **then**

**if** node is a branching node **then**

$MCRY(node.magAngle, p_q + lastOneNodes.qubits, node.qubit)$

$last1NodesL \leftarrow \{\}$

**else if** node has only the one branch **then**

$MCX(p_q + lastOneNodes.qubit, node.qubit)$ , $last1NodesL \leftarrow \{\}$

**else if** node is leaf node **then**

$MCX(p_q + lastOneNodes.qubit, node.qubit)$

$MCRY(node.normAngle, p_q + lastOneNodes.qubit, node.qubit)$

exit

**end if**

**end if**

BDD(node.left, currentPath + (node, left), last1NodesL + node)

BDD(node.right, currentPath + (node, right), last1Nodes)

**end procedure**

**for** i in 1..n **do**

$SWAP(q_i, q_{n+i})$, $H(q_{n+i})$

**end for**

# CHAPTER 4

# Experimentation

## 4.1 Questions to answer

Our experimental evaluation seeks to quantify how the proposed block encoding algorithms scale and compare them against one another and against a set of baseline block encoding algorithms, with a focus on circuit depth. We seek to answer three main questions:

- *How do the proposed block encodings compare to the baseline block encoding schemes?*

  The Topdown-inspired block encoding scheme always outperforms the baseline dense block encoding (FABLE), with a 38% decrease in depth with a 12.5% increase in the number of CNOTs overall. The best performing column block encoding, using the merge state preparation algorithm [13], has an 61% decrease in depth compared to its baseline for the example case. The BDD with forced sparsity decreases it's depth by 29% over its baseline in the example case.

- *How do the proposed block encoding schemes scale with size of input and sparsity (proportion of 0s and 1s per column of the SNP matrix) of input?*

  The dense block encodings scale quadratically with the size of the input, with no dependence on the sparsity of the input. The circuit depth of the topdown block encoding increases in depth by roughly 1.25 gates for each increase in input size squared, and the circuit depth of the FABLE block encoding increases roughly by 2 for each increase in input size squared.

  The sparse block encodings scale linearly with the sparsity of the matrix M and subquadratically with the size of the input ($O(NlogN)$). For example, the depth of the

column block encoding circuit using the merge state preparation algorithm increases by roughly 5340 gates for each additional non-two value added to each column for an input size of n=6. This increases to 9140 for n=7. This suggests an overall scaling of $O(NMlogN)$ for most of the sparse block encodings. This is mostly verified by the experimental comparisons with the baseline algorithms, with some algorithms performing worse or better but having the same overall scaling as the baseline.

- *How do the proposed block encoding schemes compare to one another? In which situations is one preferable to another?*

  Column block encodings seem to perform the best for SNP matrices, assuming that the matrix is significantly sparse per column. In the example 4.6, the best column block encoding approach overtook the topdown block encoding approach at an input size of $NxN, N = 2^9$ , so an input of 9 qubits. The topdown block encoding performs the best for lower dimension matrices (when the sparsity advantage can't be realized) and for matrices which don't meet the sparsity requirement. The sparse BDD block encoding didn't perform as well in the context of SNP matrices, showing that the BDDs for the alternating approach weren't sparse enough.

## 4.2 Experimental Setup

We evaluate the proposed algorithms on Google Colab using the CPU runtime which uses a single core hyper threaded Intel Xeon Processor @2.3Ghz.

All algorithms are implemented on top of the qiskit library [26]. We also use state preparation algorithms implemented in the qclib library [2] for column block encodings, and use the FABLE library [7] to provide an overall baseline algorithm for dense matrices.

All our code and tests can be found in the following repository: `https://github.com/Jeff848/column_block_encoding`.

### 4.2.1 Input Data

We evaluate our algorithms on generated random SNP matrices of set sizes with a set amount of sparsity per column, determined by setting the number of zeroes and ones per column. We then compare the produced circuit's depth and CNOT cost. Because the CNOT cost is largely correlated with the circuit depth, we focus our analysis mainly on the produced circuit depth.

### 4.2.2 Baseline Algorithms

We use three baseline block encoding schemes that are simple in construction and are thus easy to understand in terms of circuit depth complexity relative to the size of the input and "sparsity" of the input (ie fraction of non-majority element per column). This helps us more easily quantify the circuit depth complexity of the block encoding schemes with similar setups. These three baseline block encoding constructions correspond to the three main categories of our block encoding construction: dense block encodings, block encodings that are encoded with a column-based approach (sparsity enforced column-wise), and sparse block encodings that are encoded all at once (ie the BDD block encoding discussed previously 3). We denote this last category as sparse non-column block encodings for convenience.

- *FABLE*

  The original SOTA dense block encoding with $O(N^2)$ depth. We assume no approximation is used.

- *"Simple" column block encoding*

  We take advantage of the state preparation to block encoding reduction to construct a simple column block encoding algorithm that constructs a state using multi-control Ry gates to set the rotation qubit to encode the matrix value (see naive $O_A$ oracle implementation in FABLE [8]). This provides an easy baseline for column block encoding schemes, with $O(NMlogN + NlogN)$ depth corresponding to the depth complexity of

the CNOTs.

- *"Direct" block encoding*

  We take the above approach and skip the column check, directly use multi-control Ry gates to implement the controlled qsp oracle $O_A$. This corresponds to a sparse non-column block encoding baseline with a depth of $O(NMlogN)$.

## 4.3 Comparison of Block Encodings to Baseline Algorithms

Here, we initially compare each new block encoding construction with the baseline constructions defined previously. We evaluate our algorithms on a $N = 2^5$ $N \times N$ SNP matrix with a sparsity of two 0's and two 1's per column.

For the column based block encodings, we see that most non-sparse column based block encodings perform worse than the baseline in the results 4.1, which is to be expected. Of the tested state preparation algorithms, only the merge [13], lowrank [3], BDD [23], and baalowrank [3] (bounded approximation of the lowrank preparation) state preparation algorithms are designed to work on sparse states. Thus, they benefit from the enforced sparsity from the LCU and frequency centering techniques, and thus use less circuit depth with the added variations. In contrast, the $O(N^2)$ state preparation algorithms, such as qiskit's base state preparation algorithm which uses quantum multiplexing [27], perform worse with the enforced sparsity. We also see that the column based block encoding with the merge state preparation algorithm performs the best in the column block encoding category with the current experimental setup, with a 61% decrease in depth and 55% decrease in cnots.

For the dense block encodings 4.2, we see that the topdown block encoding utilizes around half the depth for a minimal increase in cnot count (40% decrease in depth with a 12.5% in cnots). This lines up with the expected circuit depth complexity. Sparsity does not contribute to the circuit depth for these algorithms, as the dense block encodings always attempt to encode the whole state binary tree without reduction.

For the sparse non-column block encodings 4.3, we see once again that enforcing the

| n | sparsity | method | lib | qubits | cnots | depth |
|---|---|---|---|---|---|---|
| 5 | 0.125 | simple | baseline | 12 | 10255 | 19795 |
| 5 | 0.125 | base no lcu | qiskit | 12 | 9999 | 19365 |
| 5 | 0.125 | base all variations | qiskit | 15 | 17377 | 33592 |
| 5 | 0.125 | lowrank no lcu | qclib | 12 | 8913 | 18411 |
| 5 | 0.125 | lowrank all variations | qclib | 15 | 9039 | 16445 |
| 5 | 0.125 | merge no lcu | qclib | 12 | 178186 | 364875 |
| 5 | 0.125 | merge all variations | qclib | 15 | 4609 | 7701 |
| 5 | 0.125 | svd no lcu | qclib | 12 | 10513 | 21678 |
| 5 | 0.125 | svd all variations | qclib | 15 | 17455 | 33593 |
| 5 | 0.125 | topdown no lcu | qclib | 12 | 9673 | 18355 |
| 5 | 0.125 | topdown all variations | qclib | 15 | 15617 | 28233 |
| 5 | 0.125 | baalowrank no lcu | qclib | 12 | 8913 | 18411 |
| 5 | 0.125 | baalowrank all variations | qclib | 15 | 9039 | 16445 |
| 5 | 0.125 | isometry no lcu | qclib | 12 | 9231 | 18981 |
| 5 | 0.125 | isometry all variations | qclib | 15 | 17647 | 35140 |
| 5 | 0.125 | BDD all variations | BDD, itenmc | 17 | 16864 | 28329 |

Table 4.1: Results of running various column block encodings on a $2^5 \times 2^5$ matrix with a sparsity of 4. A "no lcu" method indicates a column block encoding using just the controlled state preparation unitary of the whole state, while a "all variations" method indicates that both the lcu and frequency centering techniques were used. Lower depth is better.

| n | sparsity | method | lib | qubits | cnots | depth |
|---|---|---|---|---|---|---|
| 5 | 0.125 | fable | fable | 11 | 1039 | 2041 |
| 5 | 0.125 | topdown | columnblockenc | 11 | 1167 | 1282 |

Table 4.2: Results of dense block encodings on a $2^5 \times 2^5$ matrix with sparsity of 4

| n | sparsity | method | lib | qubits | cnots | depth |
|---|---|---|---|---|---|---|
| 5 | 0.125 | direct | columnblockenc | 12 | 29199 | 53116 |
| 5 | 0.125 | BDD sparse | columnblockenc | 14 | 21199 | 37537 |
| 5 | 0.125 | BDD no sparse | columnblockenc | 13 | 78727 | 139004 |

Table 4.3: Results of sparse non-column block encodings on a $2^5 \times 2^5$ matrix with sparsity of 4.

sparsity in the BDD is worth the sacrifice in CNOT cost due to LCU controlled unitaries. The BDD with forced sparsity improves over its baseline by 29% depth in the example case. One thing to note is that the BDD approach without enforced sparsity with LCU is significantly worse than the baseline approach, and seems to indicate that there is some inefficiency with the path commitment stage, which contributes the most to circuit depth.

## 4.4 Scaling of Block Encodings with regards to input size

We check that the improvements of the proposed block encodings are maintained for increasing input size. With the same sparsity of 4, we evaluate the block encoding algorithms on varying input matrix size from n=2 to n=6.

For the dense block encoding and sparse non-column block encoding schemes, we find that the scaling between the size of the input and the circuit depth is consistent between the algorithms and the baseline. For the dense block encodings 4.4, both algorithms produce circuits that increase in circuit depth at a similar rate of $O(N^2)$. This can be modeled with the following equations:

$$\textit{Topdown depth} = 1.24998877(2^n)^2 + 4.36565551$$

$$\textit{FABLE depth} = 1.99943658(2^n)^2 - 9.88427763$$

This matches the expected depth improvement (with the actual number being slightly lower due to CNOT overlapping)in the Topdown block encoding section. Similarly, the BDD and the direct baseline block encodings also increase in circuit depth at the same rate 4.4.

The nonsparse BDD block encoding increases at a far faster rate, indicating that the BDD block encoding has $O(NMlogN)$ scaling of depth.

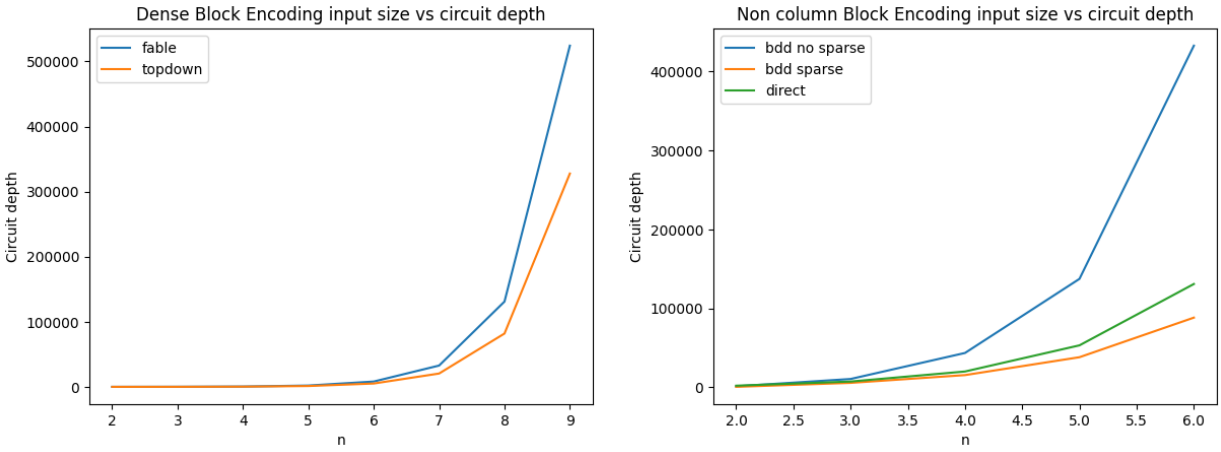


Figure 4.1: Input size vs Circuit Depth of dense and sparse non-column block encoding

The column-based block encodings 4.4 have more variation when it comes to the scaling of circuit depth with respect to input size. We specifically examine the sparse column block encodings, with the topdown column block encoding as a reference $O(N^2)$ scaling algorithm. First, we note that the lowrank column block encoding algorithm seems to scale quadratically with respect to the size of the input. This matches the worst-case scenario of the low rank state preparation algorithm [3] where it is unable to find an efficient bipartition. The likelihood of an efficient bipartition depends on a low Schmidt measure, which is not guaranteed with a sparse random state. We also notice that the circuit depth of the merge column block encoding algorithm seems to be increasing at a slower rate as compared to the baseline. This also matches with the original paper's data [13], which suggests that as long as the sparsity condition is met $o(2^n/n)$, the method is asymptotically more effective. The circuit depth of the column block encoding using BDD state preparation also increases at the same rate as the baseline block encoding.

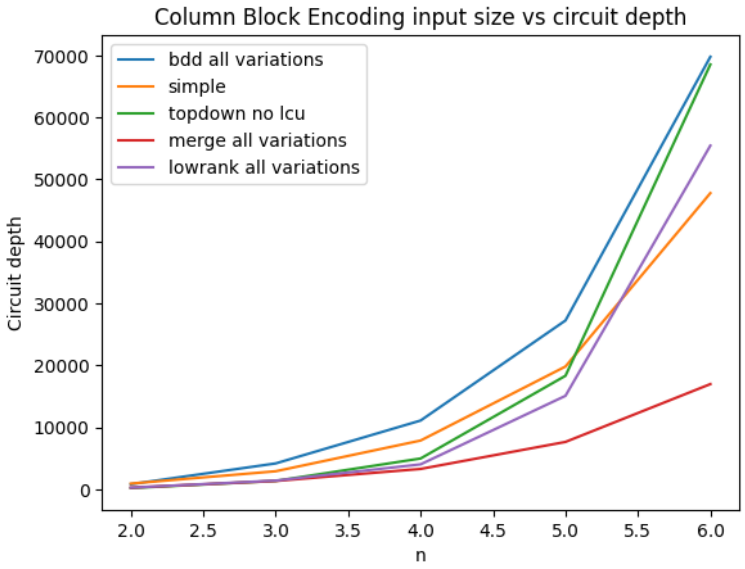


Figure 4.2: Input size vs Circuit Depth of the sparse column block encodings and the topdown column block encoding for comparison

## 4.5 Scaling of Block Encodings with regards to sparsity

We also check the scaling of the block encoding algorithms with respect to the sparsity of the input to both an n=6 and n=5 input SNP matrix. For n=5, we check up to a sparsity of 0.5 the size of the input, and for n=6 we check up to a sparsity of 0.2 of the input.

For the column block encodings, we see that all of the sparse encoding algorithms scale linearly with the sparsity of the input 4.5, 4.5. The low-depth column block encoding barely scales with respect to sparsity, but as shown previously suffers from a worst case quadratic depth with respect to the size of the input. The BDD column block encoding approach increases the most per increase in sparsity, while the simple and merge column block encodings increase in depth at roughly the same rate. Specifically, the simple and merge column block encoding depth wrt sparsity can be modeled like so for n=5:

$$Merge\ depth = 5379.625M - 14827.5$$

$$Simple\ depth = 5049.95M + 2747$$

and for n=6:

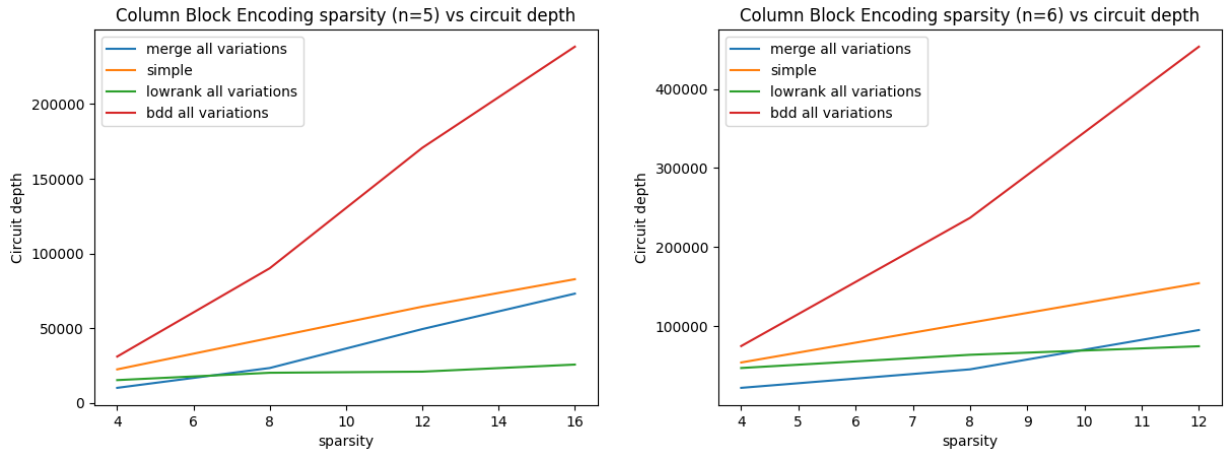


Figure 4.3: Sparsity vs circuit depth scaling for column block encodings of data matrix size n=5 and n=6 (ie $2^n \times 2^n$)

$$\textit{Merge depth} = 9140M - 18975$$

$$\textit{Simple depth} = 12541.5M + 3884.66666667$$

This seems to verify that the merge column block encoding performs the best, as its scaling with respect to sparsity increases less than the baseline block encoding as n increases. Another difference between the n=5 and n=6 experiments is the sparsity at which the sparse block encodings perform better or worse than the non-sparse block encodings. We use the low depth block encoding as reference. For n=5, that sparsity is around 6 non-two values per column, while for n=6 its around 10 non-two values per column. Once again, this matches with the idea of a required sparsity condition that depends on the size of the input in order for sparse column block encodings to get less depth than dense block encodings.

The same result is reflected in the non-column sparse block encodings 4.5, 4.5. One thing to note is that the sparse BDD circuit depth increases at a rate less than the baseline approach. Specifically, their circuit depths can be modeled like so for n=5:

$$\textit{sparse BDD depth} = 6965.425M + 10578.5$$

$$\textit{direct depth} = 17032.M - 4396.5$$

and for n=6:

$$\textit{sparse BDD depth} = 19489.75M + 9478.67$$

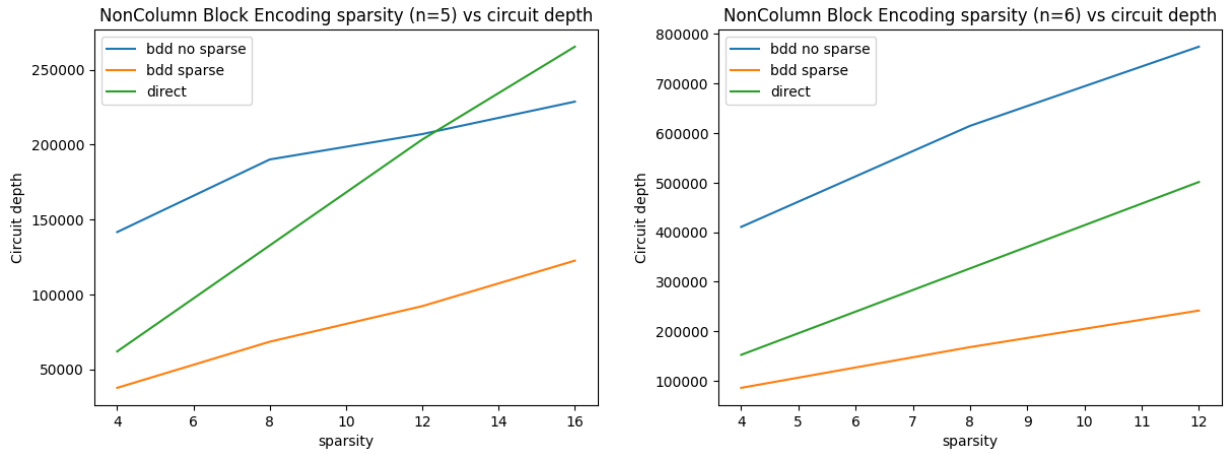


Figure 4.4: Sparsity vs circuit depth scaling for sparse non-column block encodings of data matrix size n=5 and n=6 (ie $2^n \times 2^n$)

$$direct\ depth = 43579.875M - -21713.33$$

This seems to reflect the idea that with somewhat less sparsity, the BDD encoding algorithm is able to find a more efficient BDD tree due to more overlap in the binary tree and thus encode the data more efficiently.

## 4.6 Overall evaluation

We ultimately evaluate the best block encodings of each category and directly compare their circuit depth with respect to input size for a fixed sparsity of 4.

The results 4.6 reflect the insights that we have seen with the other experiments. With a low enough sparsity, the column block encoding using the merge state preparation algorithm outperforms the topdown-block encoding at around an input size of n=9. Unfortunately, this means that for the target matrix size of $2^{18}$ by $2^{18}$, the sparsity of $M = 2^{16}$ is too high for gaining an advantage with the current sparse block encoding algorithms, as the sparsity condition:

$$4 \leq c(2^9/9); c \geq 9/2^7$$

$$M = c(2^{18}/18) = 18432$$

only seems to indicate efficient sparse block encodings for $M = 18432$ for the target matrix size. However, state preparation algorithms that trade width for circuit depth could provide a more feasible column block encoding circuit depth at greater input sizes for less sparse matrices.

The topdown block encoding remains the best block encoding for lower dimensional matrices as well as higher dimensional matrices with little to no sparsity, as it does not linearly scale with sparsity like the other block encodings. Finally, the BDD sparse block encoding did not perform well in comparison to the other block encoding algorithms. However, the smaller scaling with respect to sparsity seems to suggest that the BDD sparse block encoding would perform better in the case of a sparse/low entropy data matrix that did not have data evenly distributed across the columns, thus reducing the size of the BDD. This way, entire quadrants of the matrix can be skipped in the BDD block encoding algorithm. Perhaps with different variable orderings the BDDs could be sparse enough to compete with the other block encoding algorithms.

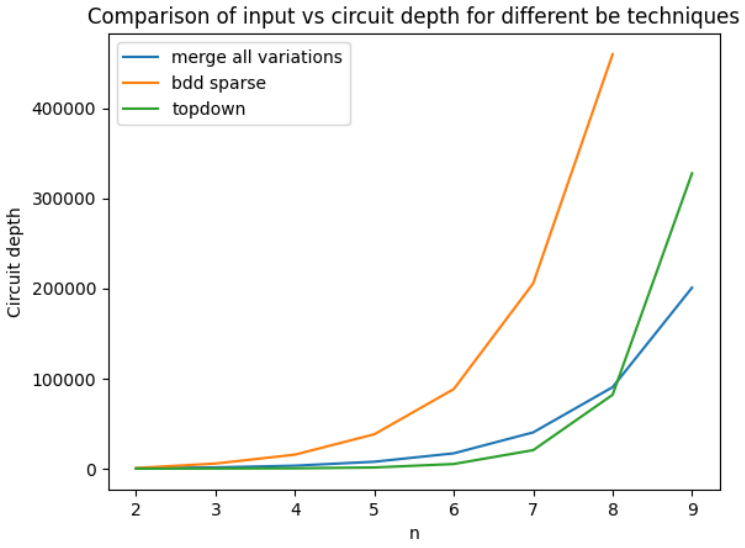


Figure 4.5: Direct comparison of the performance of the best block encodings of each category for a sparsity of 4 non-majority values per column

# CHAPTER 5

# Conclusion

In this work, we present three new block encoding algorithms motivated by a new understanding of the relations between various quantum data loading problems. Such block encodings are necessary for implementing key linear algebraic mechanisms via QSVT, which could accelerate DNA analysis of SNP matrices. We focus on block encoding algorithms that encode low density data, which can be translated to sparse data at an additional subnormalization cost.

We introduce the idea of equality between quantum data loading problems that is currently not as considered in the literature. Through reduction and a more general set of definitions, we are able to demonstrate the key idea that the block encoding problem is easily reducible to a controlled quantum state preparation problem. Similarly, we are able to show that controlled quantum state preparation can be defined as a state preparation problem, and for a subset of state preparation algorithms that we denote "recursive", we are able to treat the encoding of controlled qubits the same as regular "target" qubits of a state preparation.

Through our introduced techniques, we are able to construct a block encoding scheme that can consistently improve on the depth the state-of-the-art dense block encoding scheme FABLE, as well as introduce several sparse block encoding schemes that are able to outperform the dense block encoding schemes after a set input size, given the sparsity meets a sparsity condition. We are able to consistently outperform the baseline algorithms that we define which use simple multicontrol rotation and cnot constructions. We analyze the relationship between the circuit depth and size/sparsity of the input and determine that it

is subquadratic with respect to the input size for sparse block encoding algorithms.

**Future Steps** As mentioned previously, there seems to be several inefficiencies in the BDD block encoding algorithm. For one, the introduced subnormalization in a control node makes it so a path does not merge back into other paths, losing out on efficiency. Efficiency can be improved by allowing for an early commit of subnormalization on the rotation qubit. Additionally, there are some situations where a control qubit can be potentially treated as a non-branching node.

Another direction is to take more depth-efficient but wider recursive state preparation algorithms and apply the alternating construction technique on them. This could potentially lead to a better tradeoff between circuit depth and circuit width that is currently not studied by this paper. Similarly, one can simply take depth-efficient state preparation algorithms and apply them directly via the column based block encoding construction. Furthermore, it still remains to be seen if state preparation algorithms that are based off different divide-and-conquer strategies, such as via shortest path [30], can be converted into an alternating CQSP formulation. The research into state preparation is vast, and some state preparation algorithms not studied in this paper might be applicable to our block encoding construction.

Furthermore, additional experiments can be performed between the various block encodings in order to evaluate compilation/optimization improvement as well as accuracy cost on actual quantum computers. The proposed block encoding algorithms could possibly benefit differently from optimization and compilation, as they differ in the amount of overlapping gate operations and target qubit diversity. For example, an optimizer like VOQC [15] could find more replacement and propagate-cancel optimizations for the non-column sparse block encoding as opposed to the topdown block encoding scheme. Similarly, a compiler targeting a specific quantum architecture such as Atomique [29] might be able to find better qubit mappings/more efficient controlled QSP for the column block encoding which has a more regular qubit access pattern, reducing expensive SWAP operations and increasing gate parallelism in the process.

# APPENDIX A

# Appendix

## A.1 Iten Multicontrol Scheme

Iten's paper provides an efficient construction for multicontrol cnots with linear depth [16]. There are two main circuit contructions that are used:

**Lemma 5.** *Let $n \geq 5$ denote the total number of qubits considered and k denote the number of controls of the multicontrol cnot gate, $k \in \{1, ..., \lceil n/2 \rceil\}$. Then a multi control cnot gate $C_{k,n}(\sigma_x)$ can be implemented with at most (8k-6) CNOT gates with the following general construction:*

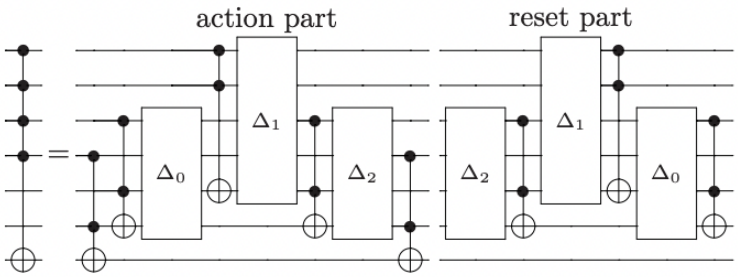


Figure A.1: Iten Multi control cnot construction

**Lemma 6.** *Let $n \geq 5$ denote the total number of qubits considered. Then a multi control cnot gate $C_{n-2,n}(\sigma_x)$ can be implemented with two $C_{k,n}(\sigma_x)$ and two $C_{n-k-1,n}(\sigma_x)$ gates for $k \in \{2, ..., n-3\}$.*

## A.2 Linear Combination of Unitaries

We use Linear Combination of Unitaries to enforce sparsity in our block encoding problems.

**Definition 6** (State Preparation Pair)**.** *Let $y \in C^m$ and $||y||_1 \leq \beta$, the pair of unitaries $(P_L, P_R)$ are called a $(\beta, n, \epsilon)$ state preparation pair of $y$ if $P_L|0^{\otimes n}\rangle = \sum_{j=0}^{2^n-1} c_j|j\rangle$ and $P_R|0^{\otimes n}\rangle = \sum_{j=0}^{2^n-1} d_j|j\rangle$ such that*

$\sum_{j=0}^{m-1} |\beta c_j^* d_j - y_j| \leq \epsilon_1$ *and* $c_j^* d_j = 0$ *for any* $j \in m, ..., 2^n - 1$

**Lemma 7** (Linear Combination of Unitaries)**.** *Let $A = \sum_{j=0}^{m-1} y_j A_j$ be a s-quibit operator where $||y||_1 \leq \beta$. Suppose $(P_L, P_R)$ is a $(\beta, n, \epsilon_1)$-state preparation pair for $y$, $W = \sum_j |j\rangle\langle j| \otimes U_j + ((I - \sum_j |0\rangle\langle 0|) \otimes I_a \otimes I_s)$ is an $(n + a + s)$ qubit unitary such that for all $j$ we have that $U_j$ is an $(\alpha, a, \epsilon_2)$-block encoding of $A_j$. Then the unitary*

$W' = (P_L^\dagger \otimes I_a \otimes I_b)W(P_R \otimes I_a \otimes I_b)$ *is a block encoding of $A$*

## A.3 Uniform Rotation Gates

We follow FABLE's approach for implementing Uniform Controlled Rotation Gates.

With a gate sequence of $2^n$ gates alternating between cnot and ry gates on a single rotation qubit, we can implement the uniform controlled rotation gate. Note that the control qubit for the $l$th CNOT gate is determined by the bit where the $l$th and the $(l+1)$th Gray code differ. That is, the gray code permuatations determine the CNOT control ordering. The angles of the circuit are related through a linear system which can be solved via a fast Walsh-Hadamard transform.

# Bibliography


[1] Aman Agrawal, Alec M. Chiu, Minh Le, Eran Halperin, and Sriram Sankararaman. Scalable probabilistic pca for large-scale genetic variation data. *bioRxiv*, 2019.

[2] Israel F. Araujo, Ismael C. S. Araújo, Leon D. da Silva, Carsten Blank, and Adenilton J. da Silva. Quantum computing library, 2 2023.

[3] Israel F. Araujo, Carsten Blank, Ismael C. S. Araújo, and Adenilton J. da Silva. Low-rank quantum state preparation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 43(1):161–170, 2024.

[4] Israel F Araujo, Daniel K Park, Teresa B Ludermir, Wilson R Oliveira, Francesco Petruccione, and Adenilton J Da Silva. Configurable sublinear circuits for quantum state preparation. *Quantum Information Processing*, 22(2):123, 2023.

[5] Clare Bycroft, Colin Freeman, Desislava Petkova, Gavin Band, Lloyd Elliott, Kevin Sharp, Allan Motyer, Damjan Vukcevic, Olivier Delaneau, Jared O'Connell, Adrian Cortes, Samantha Welsh, Alan Young, Mark Effingham, Gil McVean, Stephen Leslie, Naomi Allen, Peter Donnelly, and Jonathan Marchini. The uk biobank resource with deep phenotyping and genomic data. *Nature*, 562, 10 2018.

[6] Daan Camps, Lin Lin, Roel Van Beeumen, and Chao Yang. Explicit quantum circuits for block encodings of certain sparse matrices, 2023.

[7] Daan Camps and Roel. Van Beeumen. Fable: Fast approximate block-encodings, September 2022.

[8] Daan Camps and Roel Van Beeumen. Fable: Fast approximate quantum circuits for block-encodings. In *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, September 2022.

[9] Alec M. Chiu, Erin K. Molloy, Zilong Tan, Ameet Talwalkar, and Sriram Sankararaman. Inferring population structure in biobank-scale genomic data. *bioRxiv*, 2021.

[10] B. David Clader, Alexander M. Dalzell, Nikitas Stamatopoulos, Grant Salton, Mario Berta, and William J. Zeng. Quantum resources required to block-encode a matrix of classical data. *IEEE Transactions on Quantum Engineering*, 3:1–23, 2022.

[11] Yuxuan Du, Tongliang Liu, Yinan Li, Runyao Duan, and Dacheng Tao. Quantum divide-and-conquer anchoring for separable non-negative matrix factorization, 2018.

[12] András Gilyén, Yuan Su, Guang Hao Low, and Nathan Wiebe. Quantum singular value transformation and beyond: exponential improvements for quantum matrix arithmetics. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC '19. ACM, June 2019.

[13] Niels Gleinig and Torsten Hoefler. An efficient algorithm for sparse quantum state preparation. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 433–438, 2021.

[14] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical Review Letters*, 103(15), October 2009.

[15] Kesha Hietala, Robert Rand, Shih-Han Hung, Xiaodi Wu, and Michael Hicks. A verified optimizer for quantum circuits. *Proceedings of the ACM on Programming Languages*, 5(POPL):1–29, January 2021.

[16] Raban Iten, Roger Colbeck, Ivan Kukuljan, Jonathan Home, and Matthias Christandl. Quantum circuits for isometries. *Physical Review A*, 93(3), March 2016.

[17] Linda Kachuri, Thomas J. Hoffmann, Yu Jiang, Sonja I. Berndt, John P. Shelley, Kerry Schaffer, Mitchell J. Machiela, Neal D. Freedman, Wen-Yi Huang, Shengchao A. Li, Ryder Easterlin, Phyllis J. Goodman, Cathee Till, Ian Thompson, Hans Lilja, Stephen K. Van Den Eeden, Stephen J. Chanock, Christopher A. Haiman, David V. Conti, Robert J. Klein, Jonathan D. Mosley, Rebecca E. Graff, and John S. Witte. Genetically adjusted psa levels for prostate cancer screening. *medRxiv*, 2023.

[18] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum principal component analysis. *Nature Physics*, 10(9):631–633, July 2014.

[19] Guang Hao Low and Isaac L. Chuang. Optimal hamiltonian simulation by quantum signal processing. *Physical Review Letters*, 118(1), January 2017.

[20] Guang Hao Low and Isaac L. Chuang. Hamiltonian simulation by qubitization. *Quantum*, 3:163, July 2019.

[21] Alessandro Luongo and Changpeng Shao. Quantum algorithms for spectral sums, 2020.

[22] John M. Martyn, Zane M. Rossi, Andrew K. Tan, and Isaac L. Chuang. Grand unification of quantum algorithms. *PRX Quantum*, 2(4), December 2021.

[23] Fereshte Mozafari, Giovanni De Micheli, and Yuxiang Yang. Efficient deterministic preparation of quantum states using decision diagrams. *Physical Review A*, 106(2), August 2022.

[24] Quynh T. Nguyen, Bobak T. Kiani, and Seth Lloyd. Block-encoding dense and full-rank kernels using hierarchical matrices: applications in quantum numerical linear algebra. *Quantum*, 6:876, December 2022.

[25] Ali Pazokitoroudi, Yue Wu, Kathryn S. Burch, Kangcheng Hou, Aaron Zhou, Bogdan Pasaniuc, and Sriram Sankararaman. Efficient variance components analysis across millions of genomes. *bioRxiv*, 2020.

[26] IBM Research. Qiskit api reference. https://docs.quantum.ibm.com/api/qiskit, 2017–2024.

[27] V.V. Shende, S.S. Bullock, and I.L. Markov. Synthesis of quantum-logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(6):1000–1010, June 2006.

[28] Christoph Sünderhauf, Earl Campbell, and Joan Camps. Block-encoding structured matrices for data input in quantum computing. *Quantum*, 8:1226, January 2024.

[29] Hanrui Wang, Pengyu Liu, Daniel Bochen Tan, Yilian Liu, Jiaqi Gu, David Z. Pan, Jason Cong, Umut A. Acar, and Song Han. Atomique: A quantum compiler for reconfigurable neutral atom arrays, 2024.

[30] Hanyu Wang, Bochen Tan, Jason Cong, and Giovanni De Micheli. Quantum state preparation using an exact cnot synthesis formulation, 2024.

[31] Pei Yuan and Shengyu Zhang. Optimal (controlled) quantum state preparation and improved unitary synthesis by quantum circuits with any number of ancillary qubits. *Quantum*, 7:956, March 2023.