How to Compile Some
NAND Formula Evaluators

Robert R. Tucci
P.O. Box 226
Bedford, MA 01730
tucci@ar-tiste.com

February 5, 2008

Abstract

We say a unitary operator acting on a set of qubits has been compiled if it has been expressed as a SEO (sequence of elementary operations, like CNOTs and single-qubit operations). SEO’s are often represented as quantum circuits. [arXiv:quant-ph/0702144] by Farhi-Goldstone-Gutmann has inspired a recent flurry of papers, that propose quantum algorithms for evaluating NAND formulas via quantum walks over tree graphs. These algorithms use two types of unitary evolution: oracle and non-oracle. Non-oracle evolutions are independent of the NAND formula input, whereas oracle evolutions depend on this input. In this paper we compile (i.e., give explicit SEOs and their associated quantum circuits for) the oracle and non-oracle evolution operators used in some of these NAND formula evaluators. We consider here only the case of balanced binary NAND trees. Our compilation methods are based on the CSD (Cosine Sine Decomposition), a matrix decomposition from Linear Algebra. The CS decomposition has been used very successfully in the past to compile unstructured unitary matrices exactly.
1 Introduction and Motivation

(The picture on the front page is an incidence matrix for a binary tree. Empty squares indicate zero entries and bullets indicate non-zero entries.)

Recently, Farhi-Goldstone-Gutmann proposed in FGG07 [1] a quantum algorithm for evaluating certain boolean formulas \( \phi : \{0, 1\}^{N_{lvs}} \to \{0, 1\} \) on \( N_{lvs} \) variables. FGG07 considers only those \( \phi() \) representable as balanced binary NAND trees. These are perfectly bifurcating trees, with a NAND at every non-leaf node, with \( N_{lvs} \) leaves and \( \log_2(N_{lvs}) \) levels, where each leaf represents a different input variable of \( \phi() \). The FGG07 algorithm evaluates these balanced binary NAND formulas in time \( \mathcal{O}(\sqrt{N_{lvs}}) \).

The FGG07 algorithm uses a continuous time quantum walk. The input to the NAND formula is entered into the quantum walk by means of an “oracle Hamiltonian”, which is applied constantly and continuously during the quantum walk. One describes this situation as a **continuous querying of the oracle**.

A few days after FGG07 appeared at ArXiv, Cleve et al published Cle07 [2] at the same library. This short two page paper points out that one can translate the FGG07 algorithm to a counterpart algorithm. The counterpart algorithm enters the input into the quantum walk during a finite number of interruptions or queries. This alternative method of entering the input is described as a **discrete querying of the oracle**. Cle07 shows that the counterpart algorithm will yield a solution to a balanced binary NAND formula after \( \mathcal{O}((N_{lvs})^{\frac{1}{2}+\epsilon}) \) queries, for any \( \epsilon > 0 \).

Next, Chi07 [3], by Childs et al, appeared on ArXiv. Chi07 proposes an FGG07-inspired, discrete-queries algorithm for evaluating arbitrary (not necessarily balanced or binary) NAND formulas. The Chi07 algorithm, like the Cle07 one, requires \( \mathcal{O}((N_{lvs})^{\frac{1}{2}+\epsilon}) \) queries to evaluate a NAND formula. It uses a quantum walk on a tree graph, with a tail of nodes attached to the root node of the tree. In contrast, the FGG07 algorithm attaches a runway to the tree. The runway is a finite line of nodes attached at its middle node to the root node of the tree. The Chi07 algorithm uses quantum phase estimation to deduce the value of the NAND formula. In contrast, FGG07 uses a scattering experiment to deduce this value.

Next, Amb07 [4], by Ambainis, appeared on ArXiv. Amb07 describes an FGG07-inspired, discrete-queries algorithm that can evaluate balanced binary NAND formulas with \( \mathcal{O}((N_{lvs})^{\frac{1}{2}}) \) queries. Thus, the Amb07 algorithm improves over Cle07 and Chi07 by removing the \( \epsilon \). Amb07 also gives an algorithm that allows one to evaluate arbitrary NAND formulas with \( \mathcal{O}((N_{lvs})^{\frac{1}{2}+\mathcal{O}(\sqrt{\log_2 N_{lvs}})}) \) queries.

We say a unitary operator acting on a set of qubits has been compiled if it has been expressed as a SEO (sequence of elementary operations, like CNOTs and single qubit operations). SEO’s are often represented as quantum circuits.

This paper, like Cle07, Chi07 and Amb07, considers FSS07-inspired algorithms with discrete querying. Such algorithms use two types of unitary evolution: oracle and non-oracle. Non-oracle evolutions are independent of the NAND formula input, whereas oracle evolutions depend on this input. The goal of this paper is to compile
these oracle and non-oracle evolutions. We present explicit SEO’s and their associated circuits, for these evolutions. Such circuits will be required for most physical implementations of the algorithms proposed in Cle07, Chi07 and Amb07, but are not given by those papers. Such missing circuits constitute a large gap in our practical knowledge that this paper is intended to fill.

Some papers (for example, Refs. [5, 6, 7, 8]) have previously addressed the issue of compiling quantum walks over trees and other graphs. Some of these papers use oracles to encode the graph topology. Our approach is different. We do not use any oracles to encode the graph topology.

This paper considers only balanced binary NAND trees, although I think some of the methods of this paper can be applied to more general trees.

Our compilation methods are based on the CSD (Cosine Sine Decomposition) [9], a matrix decomposition from Linear Algebra. This decomposition is explained in Section 3.4. For now, suffice it to say that the CS decomposition has been used very successfully in the past to compile unstructured unitary matrices exactly.

### 2 Notation

In this section, we will define some notation that is used throughout this paper. For additional information about our notation, we recommend that the reader consult Ref. [10]. Ref. [10] is a review article, written by the author of this paper, which uses the same notation as this paper.

We will often use the symbol $N_B$ for the number ($\geq 1$) of qubits and $N_S = 2^{N_B}$ for the number of states with $N_B$ qubits. The quantum computing literature often uses $n$ for $N_B$ and $N$ for $N_S$, but we will avoid this notation. We prefer to use $n$ for the number operator, defined below.

Let $\text{Bool} = \{0, 1\}$. As usual, let $\mathbb{Z}, \mathbb{R}, \mathbb{C}$ represent the set of integers (negative and non-negative), real numbers, and complex numbers, respectively. For integers $a, b$ such that $a \leq b$, let $Z_{a,b} = \{a, a+1, \ldots b-1, b\}$. For $\Gamma$ equal to $\mathbb{Z}$ or $\mathbb{R}$, let $\Gamma^{>0}$ and $\Gamma^{\geq0}$ represent the set of positive and non-negative $\Gamma$ numbers, respectively. For any positive integer $k$ and any set $S$, let $S^k$ denote the Cartesian product of $k$ copies of $S$; i.e., the set of all $k$-tuples of elements of $S$.

We will use $\theta(S)$ to represent the “truth function”; $\theta(S)$ equals 1 if statement $S$ is true and 0 if $S$ is false. For example, the Kronecker delta function is defined by $\delta_{xy} = \delta(x, y) = \theta(x = y)$.

Let $\mathbf{0} = 1$ and $\mathbf{1} = 0$. If $\vec{a} = a_{N_B-1} \ldots a_2 a_1 a_0$, where $a_\mu \in \text{Bool}$, then $\text{dec}(\vec{a}) = \sum_{\mu=0}^{N_B-1} 2^\mu a_\mu = a$. Conversely, $\vec{a} = \text{bin}(a)$.

We define the single-qubit states $|0\rangle$ and $|1\rangle$ by

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \quad (1)$$

If $\vec{a} \in \text{Bool}^{N_B}$, we define the $N_B$-qubit state $|\vec{a}\rangle$ as the following tensor product
\[ |\vec{a}\rangle = |a_{NB-1}\rangle \otimes \ldots \otimes |a_1\rangle \otimes |a_0\rangle . \] (2)

For example,

\[ |01\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} . \] (3)

When we write a matrix, and leave some of its entries blank, those blank entries should be interpreted as zeros. $I_k$ and $0_k$ will represent the $k \times k$ unit and zero matrices, respectively. For any matrix $A \in \mathbb{C}^{p \times q}$, $A^*$ will stand for its complex conjugate, $A^T$ for its transpose, and $A^\dagger$ for its Hermitian conjugate. For any two same-sized square matrices $A$ and $B$, we define the o-dot product $\odot$ by $A \odot B = ABA^\dagger$.

For any matrix $A$ and positive integer $k$, let

\[ A^{\otimes k} = \overbrace{A \otimes \cdots \otimes A}^{k \text{ copies of } A} , \] (4)

\[ A^{\oplus k} = \overbrace{A \oplus \cdots \oplus A}^{k \text{ copies of } A} . \] (5)

Suppose $\beta \in \mathbb{Z}_{0,N_B-1}$ and $M$ is any $2 \times 2$ matrix. We define $M(\beta)$ by

\[ M(\beta) = I_2 \otimes \cdots \otimes I_2 \otimes M \otimes I_2 \otimes \cdots \otimes I_2 , \] (6)

where the matrix $M$ on the right hand side is located at qubit position $\beta$ in the tensor product of $N_B$ $2 \times 2$ matrices. The numbers that label qubit positions in the tensor product increase from right to left (←), and the rightmost qubit is taken to be at position 0.

The Pauli matrices are

\[ \sigma_X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} , \quad \sigma_Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} , \quad \sigma_Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} . \] (7)

Let $\vec{\sigma} = (\sigma_X, \sigma_Y, \sigma_Z)$. For any $\vec{a} \in \mathbb{R}^3$, let $\sigma_{\vec{a}} = \vec{\sigma} \cdot \vec{a}$.

The one-qubit Hadamard matrix $H$ is defined as:

\[ H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} . \] (8)

The $N_B$-qubit Hadamard matrix is defined as $H^{\otimes N_B}$.

The number operator $n$ for a single qubit is defined by

\[ n = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} = \frac{1 - \sigma_Z}{2} . \] (9)
Note that
\[ n|0\rangle = 0|0\rangle = 0 \quad , \quad n|1\rangle = 1|1\rangle . \] (10)

We will often use \( \pi \) as shorthand for
\[ \pi = 1 - n = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} = \frac{1 + \sigma_Z}{2} . \] (11)

Define \( P_0 \) and \( P_1 \) by
\[ P_0 = \pi = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} = |0\rangle\langle 0| \quad , \quad P_1 = n = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} = |1\rangle\langle 1|. \] (12)

Two other related \( 2 \times 2 \) matrices are
\[ \sigma_- = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} = |1\rangle\langle 0| \quad , \quad \sigma_+ = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} = |0\rangle\langle 1|. \] (13)

\( P_0 \) and \( P_1 \) are orthogonal projection operators and they add to one:
\[ P_a P_b = \delta(a,b) P_b \quad \text{for } a, b \in \text{Bool} , \] (14)
\[ P_0 + P_1 = I_2 . \] (15)

For \( \vec{a} \in \text{Bool}^{NB} \), let
\[ P_{\vec{a}} = P_{a_{NB-1}} \otimes \cdots \otimes P_{a_2} \otimes P_{a_1} \otimes P_{a_0} . \] (16)

For example, with 2 qubits we have
\[ P_{00} = P_0 \otimes P_0 = \text{diag}(1,0,0,0) , \] (17)
\[ P_{01} = P_0 \otimes P_1 = \text{diag}(0,1,0,0) , \] (18)
\[ P_{10} = P_1 \otimes P_0 = \text{diag}(0,0,1,0) , \] (19)
\[ P_{11} = P_1 \otimes P_1 = \text{diag}(0,0,0,1) . \] (20)

Note that
\[ P_{\vec{a}} P_{\vec{b}} = \delta(\vec{a},\vec{b}) P_{\vec{b}} \quad \text{for } \vec{a}, \vec{b} \in \text{Bool}^{NB} , \] (21)
\[ \sum_{\vec{a} \in \text{Bool}^{NB}} P_{\vec{a}} = I_2 \otimes I_2 \otimes \cdots \otimes I_2 = I_{2^{NB}} . \] (22)
If $\vec{a} \in \text{Bool}^N$, and $\beta \in \mathbb{Z}_{0,N-1}$, then we will denote $\sum_{a,\beta \in \text{Bool}} P_{\vec{a}}$ by replacing the symbol $a_\beta$ in $P_{\vec{a}}$ by a dot. For example, $\sum_{a_1 \in \text{Bool}} P_{a_2,a_1,a_0} = P_{a_1,a_0}$.

Next we explain our circuit diagram notation. We label single qubits (or qubit positions) by a Greek letter or by an integer. When we use integers, the topmost qubit wire is 0, the next one down is 1, then 2, etc. \textit{Note that in our circuit diagrams, time flows from the right to the left of the diagram.} Careful: Many workers in Quantum Computing draw their diagrams so that time flows from left to right. We eschew their convention because it forces one to reverse the order of the operators every time one wishes to convert between a circuit diagram and its algebraic equivalent in Dirac notation.

$h.c.$ will stand for Hermitian conjugate. For example, $A + h.c.$ will denote $A + A^\dagger$, and $\begin{bmatrix} A & h.c. \end{bmatrix}$ will denote $\begin{bmatrix} A & A^\dagger \end{bmatrix}$. This notation is useful when $A$ is a long expression that we do not wish to repeat.

SVD will stand for Singular Value Decomposition. An SVD of a matrix $A$ consists of matrices $U, V, \Delta$ such that $A = U\Delta V^\dagger$, where the matrices $U, V$ are unitary and $\Delta$ is a non-negative diagonal matrix. We will say that this SVD is “one-sided” if one “side” equals the identity matrix; i.e., if $U = 1$ or $V = 1$.

When we use $\prod_{i=1,3,2} Q_i$ for non-commuting operators $Q_i$, we mean $Q_1Q_3Q_2$, not $Q_1Q_2Q_3$ or any other permutation. Thus, the multiplication must be performed in the order indicated.

3 Review of Some Prerequisites

3.1 Circulant Matrices

In this section, we will review some properties of circulant matrices\cite{III}. These properties will be used later on, to compile the evolution operator for a random walk on a closed loop.

A circulant matrix is any matrix of the form

$$C = \begin{bmatrix} c_0 & c_1 & c_2 & \cdots & c_{n-2} & c_{n-1} \\ c_{n-1} & c_0 & c_1 & \cdots & c_{n-3} & c_{n-2} \\ c_{n-2} & c_{n-1} & c_0 & \cdots & c_{n-4} & c_{n-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ c_2 & c_3 & c_4 & \cdots & c_0 & c_1 \\ c_1 & c_2 & c_3 & \cdots & c_{n-1} & c_0 \end{bmatrix} \quad \text{for } n = 1, 2, 3, \ldots \text{ and } c_j \in \mathbb{C}.$$ \hspace{1cm} (23)

for $n = 1, 2, 3, \ldots$ and $c_j \in \mathbb{C}$. Note that each row of $C$ is a cyclic shift of the row above it.

If we denote the eigenvalues and eigenvectors of $C$ by

$$Cv = \lambda v,$$ \hspace{1cm} (24)
then it is easy to show (see Ref.[11]) that the $n$ eigenvalues and corresponding eigenvectors of $C$ are given by

$$\lambda_m = \sum_{k=0}^{n-1} c_k (\omega^m)^k, \quad v^{(m)} = \begin{bmatrix} 1 \\ \omega^m \\ (\omega^m)^2 \\ \vdots \\ (\omega^m)^{n-1} \end{bmatrix}, \quad \text{where} \quad \omega = e^{-\frac{2\pi i}{n}} \quad (25)$$

for $m \in \mathbb{Z}_{0,n-1}$. Define matrices $U$ and $D$ by

$$U = (v^{(0)}, v^{(1)}, v^{(2)}, \ldots, v^{(n-1)}), \quad D = \text{diag}(\lambda_0, \lambda_1, \lambda_2, \ldots, \lambda_{n-1}) \quad (26)$$

Then

$$CU = UD, \quad C = U D U^\dagger \quad (27)$$

The matrix $U$ is in fact the discrete Fourier transform (DFT) matrix.

In this paper, we will consider the case that $c_1 = c_{n-1} = g$ and all other $c_k$ are zero. In this case,

$$\lambda_m = g(\omega^m + (\omega^m)^{n-1}) = g(\omega^m + \omega^{-m}) = g 2 \cos \left( \frac{2\pi m}{n} \right) \quad (28)$$

3.2 Trotter Rescaling, Lie and Suzuki Approximants

In this section, we will review the Trotter rescaling of the Lie and Suzuki approximants of $e^{A+B}$. More information about this topic may be found in Ref.[12].

Suppose $A, B \in \mathbb{C}^{n \times n}$ and $t \in \mathbb{R}$. Define

$$L_1(t) = e^{tA} e^{tB} \quad (29)$$

In this paper, we will refer to $L_1(t)$ as the **Lie first-order approximant** of $e^{t(A+B)}$. We call it a first-order approximant because, for small $t$, according to the Baker-Campbell-Hausdorff expansion[13, 14],

$$L_1(t) = e^{t(A+B) + \frac{t^2}{2} [A,B] + O(t^3)} = e^{t(A+B)} + O(t^2) \quad (30)$$

But what if $t$ isn’t small? Even when $t$ is not small, one can still use the Lie
approximant to approximate $e^{t(A+B)}$. Indeed, if $N$ is a very large integer, then

$$L_N^N \left( \frac{t}{N} \right) = \left( e^{\frac{t}{N}A} e^{\frac{t}{N}B} \right)^N$$

$$= \left( e^{\frac{t}{N}(A+B) + \frac{t^2}{2N^2}[A,B] + \mathcal{O}(\frac{t^3}{N^2})} \right)^N$$

$$= e^{t(A+B) + \frac{t^2}{2N}[A,B] + \mathcal{O}(\frac{t^3}{N^2})}$$

$$= e^{t(A+B)} + \mathcal{O}(\frac{t^2}{N}).$$

Henceforth, will refer to this nice trick as a **Trotter rescaling** of an approximant (in this case, the Lie approximant). See Fig.1.

![Figure 1: Lie algebra “Physicist’s picture” of Trotter rescaling. The system moves from 0 to $A + B$, by moving in small increments in the $A$ and $B$ directions.](image)

Next define

$$S_2(t) = e^{\frac{t}{2}} e^{tB} e^{\frac{t}{2}}.$$

We will refer to $S_2(t)$ as the **Suzuki second-order approximant**. One can show \(13\) that for small $t$:

$$S_2(t) = e^{t(A+B) + \frac{t^3}{3N^2}[4A+2B,A,B] + \mathcal{O}(t^5)} = e^{t(A+B)} + \mathcal{O}(t^3).$$

Suzuki also defined higher order approximants based on $S_2(t)$. For $k = 1, 2, 3, \ldots$, define the **Suzuki (2$k$ + 2)th-order approximant** $S_{2k+2}(t)$ by

$$S_{2k+2}(t) = S_{2k}^2(a_{2k}t) S_{2k}((1 - 4a_{2k})t)) S_{2k}^2(a_{2k}t),$$

for some $a_{2k} \in \mathbb{R}$. It is possible to show that for $k = 1, 2, 3, \ldots$ and small $t$:

$$S_{2k+2}(t) = e^{t(A+B)} + \mathcal{O}(t^{2k+3}), \text{ if } a_{2k} = \frac{1}{4 - 4\pi + \pi}.$$
$(a_{2k})_{k=1,2,3,...}$ is a monotone decreasing sequence with $a_2 = 0.4145...$ and $\lim_{k \to \infty} a_{2k} = 1/3$.

As with the Lie approximant, it is possible to do a Trotter rescaling of the Suzuki approximants. One finds that for $k = 1, 2, 3, \ldots$, large $N$ and fixed $t$:

$$S_{2k}^N\left(\frac{t}{N}\right) = e^{i(A+B)} + O\left(\frac{t^{2k+1}}{N^{2k}}\right). \quad (36)$$

Henceforth, what we have called $N$ so far in this section will be renamed $N_T$, to distinguish it from all the other $N$’s used in this paper. The $T$ in $N_T$ stands for Trotter, since it represents the number of Trotter time slices (trots?).

If we call $N_{\text{exp}}$ the number of factors of the type $e^{\tau A}$ or $e^{\tau B}$ for some $\tau \in \mathbb{R}$, then we can compare $N_{\text{exp}}$ and the error for the Trotterized Lie and Suzuki approximants. Let

| $L_1^{N_T}(\frac{t}{N_T})$ | $N_{\text{exp}}$ | error |
|--------------------------|------------------|-------|
| $S_{2k}^{N_T}(\frac{t}{N_T})$ | $2N_T$ | $O\left(\frac{t}{N_T}\right)$ |
| $2(5)^k - 1$ | $2N_T + 1$ | $O\left(\frac{t^{2k+1}}{N^{2k}}\right)$ |

Let $N_{\text{exp}}^{T-S}$ stand for the number of exponentials in the Trotterized Suzuki approximant. Define

$$\epsilon = \frac{t^{2k+1}}{N^{2k}}. \quad (37)$$

Eq. (37) for $\epsilon$ can be inverted to get $N_T = \frac{t^{2k+1}}{\epsilon^{\frac{1}{2k}}}$. This expression for $N_T$ as a function of $\epsilon$ can be substituted into the value for $N_{\text{exp}}^{T-S}$ given in the above table, to get:

$$N_{\text{exp}}^{T-S} = 2(5)^k - 1 + \frac{1 + \frac{1}{2k}}{\epsilon^{\frac{1}{2k}}} + 1. \quad (38)$$

The previous expression for $N_{\text{exp}}^{T-S}$ has a minimum for small $\epsilon$. For example, suppose $\epsilon = 10^{-12}$. For $k = 1$, $N_{\text{exp}}^{T-S} \sim \epsilon^{-\frac{1}{2}} \approx 10^6$. For $k = 2$, $N_{\text{exp}}^{T-S} \sim \epsilon^{-\frac{1}{2k}} = 10^3$ As $k \to \infty$, $N_{\text{exp}}^{T-S} \to 2(5)^{k-1}t \to \infty$. It is possible to use calculus to obtain the value of $k$ that minimizes $N_{\text{exp}}^{T-S}$. One finds

$$k \approx \sqrt{\frac{\ln(\frac{1}{2})}{2\ln(5)}}, \quad N_{\text{exp}}^{T-S} \approx \frac{2t}{5}e^{\frac{2\ln(5)\ln(\frac{1}{2})}{2}}. \quad (39)$$

1. If $N_{\text{exp}}(S_{2k})$ is the number of exponentials in $S_{2k}$, then $N_{\text{exp}}(S_2) = 3$, $N_{\text{exp}}(S_4) = 11$, $N_{\text{exp}}(S_6) = 51$. In general, $N_{\text{exp}}(S_{2k+2}) = N_{\text{exp}}(S_{2k})5 - 4$ for $k = 1, 2, \ldots$. Solving the difference equation $f(k+1) = f(k)5 - 4$ with $f(1) = 3$ yields $f(k) = 2(5)^{k-1} + 1$. Finally, note that $N_{\text{exp}}^{T-S} = f(k)N_T - (N_T - 1)$.

2. Ref. [?] does a similar minimization for a more general Suzuki approximant that approximates $e^{i(\Lambda_1 + \Lambda_2 + \ldots + \Lambda_m)}$ where $m \geq 2$. We will only use $m = 2$ in this paper.
3.3 Multiply Controlled NOTs

In this section, we will review how to compile multiply controlled \( U(2) \) operators. More information about this topic may be found in Ref. [15].

Suppose \( U \in U(2) \). If \( \tau \) and \( \kappa \) are two different qubit positions, gate \( U(\tau)^{n(\kappa)} \) (or \( U(\tau)^{P(\kappa)} \)) is called a controlled \( U \) with target \( \tau \) and control \( \kappa \). When \( U = \sigma_X \), this reduces to a CNOT (controlled NOT). If \( \tau, \kappa_1, \kappa_2 \) are 3 different qubit positions, \( \sigma_X(\tau)^{n(\kappa_1)n(\kappa_2)} \) is called a Toffoli gate with target \( \tau \) and controls \( \kappa_1, \kappa_2 \).

Suppose \( N_K \geq 2 \) is an integer and \( \vec{b} \in \text{Bool}^{N_K} \). Suppose \( \tau, \kappa_{N_K-1}, \kappa_{N_K-2}, \ldots, \kappa_1, \kappa_0 \) are distinct qubits and \( \vec{\kappa} = (\kappa_{N_K-1}, \kappa_{N_K-2}, \ldots, \kappa_1, \kappa_0) \). Gate \( U(\tau)^{P \vec{b}(\vec{\kappa})} \) is called a multiply controlled \( U \) with target \( \tau \) and \( N_K \) controls \( \vec{\kappa} \). When \( U = \sigma_X \), this reduces to an MCNOT (multiply controlled NOT).

Our goal in this section is to show a reasonably efficient way of compiling any multiply controlled \( U(2) \) operator; i.e., to express it as a SEO consisting of CNOTs and single-qubit operations.

For any \( U \in U(2) \), a multiply controlled \( U \) can be expressed in terms of MCNOTs as follows:

\[
|0\rangle U = |0\rangle U. \tag{40}
\]

This requires one ancilla qubit in state \(|0\rangle\). The ancilla is reusable.

Remember that our goal is to show how to compile any multiply controlled \( U(2) \) operator. Now all we need to show is how to compile MCNOTs. MCNOTs can be expressed in terms of Toffoli gates as follows:

\[
|0\rangle \times |0\rangle \times |0\rangle \times |0\rangle \times |0\rangle \times |0\rangle \times |0\rangle \times |0\rangle \tag{41}
\]

For \( N_K \geq 3 \) controls, this requires \( N_K - 2 \) ancilla qubits in state \(|0\rangle\). The ancillas are reusable.

So now all we need to show is how to compile a Toffoli gate. The following identity, true for any \( V \in U(2) \), is useful in this regard:
When \( V = \sigma_X \), this becomes:

\[
\begin{array}{c}
\sigma_X \\
\sigma_X \frac{1}{2} \\
\sigma_X \frac{1}{2} \\
\sigma_X \frac{1}{2}
\end{array}
= \quad
\begin{array}{c}
\sigma_X \\
\sigma_X \frac{1}{2} \\
\sigma_X \frac{1}{2} \\
\sigma_X \frac{1}{2}
\end{array}.
\]

(43)

Thus, we can compile a Toffoli gate if we can compile a singly controlled \((\sigma_X)^{\frac{1}{2}}\). A singly controlled \(U(2)\) operator also arose in Eq. (40). So we should show how to compile a general singly controlled \(U(2)\) operator. This can be done using the following quaternion trick\[16\]. Observe that an arbitrary \(U \in U(2)\) need not have unit determinant. For example, \(\sigma_X\) and \((\sigma_X)^{\frac{1}{2}}\) don’t. So we need to express \(U\) as a product of an \(SU(2)\) matrix and a phase:

\[
U = e^{i\phi} \sigma \hat{a} \sigma \hat{b}.
\]

(44)

Let \(R\) and \(S\) be \(SU(2)\) matrices that rotate \(\hat{a}\) and \(\hat{b}\) into the x axis:

\[
\sigma \hat{a} = R \sigma_X R^\dagger, \quad \sigma \hat{b} = S \sigma_X S^\dagger.
\]

(45)

Then

\[
\begin{array}{c}
R \sigma_X R^\dagger \\
R \sigma_X R^\dagger S \\
S^\dagger \\
\end{array}
= \quad \begin{array}{c}
U(\beta)^{n(\alpha)} \\
e^{i\phi n(\alpha)} \sigma \hat{a} \sigma \hat{b} (\beta)^{n(\alpha)} \\
e^{i\phi n(\alpha)} \\
\end{array}.
\]

(46c)

Thus, we have achieved our goal of showing how to compile any multiply controlled \(U(2)\) operator.

Let \(G\) be any multiply controlled \(U(2)\) gate with \(N_K\) controls, where \(N_K \geq 1\). A consequence of the results of this section is that \(G\) can always be expressed as a SEO with \(O(N_K)\) CNOTs.
3.4 CS Decomposition

In this section, we will define the CSD (Cosine Sine Decomposition)\textsuperscript{[9]} and review how it has been applied to quantum computing prior to this paper.

Suppose that $U$ is an $N \times N$ unitary matrix, where $N$ is an even number. The CSD Theorem states\textsuperscript{[9]} that one can always express $U$ in the form

$$U = \begin{bmatrix} L_0 & 0 \\ 0 & L_1 \end{bmatrix} D \begin{bmatrix} R_0 & 0 \\ 0 & R_1 \end{bmatrix},$$

where the left and right matrices $L_0, L_1, R_0, R_1$ are $\frac{N}{2} \times \frac{N}{2}$ unitary matrices, and

$$D = \begin{bmatrix} D_{00} & D_{01} \\ D_{10} & D_{11} \end{bmatrix},$$

$$D_{00} = D_{11} = \text{diag}(C_1, C_2, \ldots, C_{\frac{N}{2}}),$$

$$D_{01} = \text{diag}(S_1, S_2, \ldots, S_{\frac{N}{2}}), \quad D_{10} = -D_{01}.$$

For all $i \in Z_{\frac{N}{2}}$, $C_i = \cos \theta_i$ and $S_i = \sin \theta_i$ for some angle $\theta_i$. Eqs.(47) can be expressed more succinctly as

$$U = (L_0 \oplus L_1)e^{i\sigma Y \otimes \Theta}(R_0 \oplus R_1),$$

where $\Theta = \text{diag}(\theta_1, \theta_2, \ldots, \theta_{\frac{N}{2}})$.

![Figure 2: Diagrammatic representation of CS decomposition, Eq.(47).](image)

Fig.2 is a diagrammatic representation of the CSD. Note that: (1)Matrix $U$ is assigned to the incoming arrow. (2)Matrix $D$ is assigned to the node. (3)Matrices $R_0 \oplus R_1$ and $L_0 \oplus L_1$ are each assigned to an outgoing arrow.

The CS decomposition was first used for quantum compiling in Tuc99\textsuperscript{[17]}. In the Tuc99 compiling algorithm, the CSD is used recursively. A nice way of picturing this recursive use of the CSD is to represent each CSD application by a node, as

\textsuperscript{3}Actually, this is only a special case of the CSD Theorem—the case which is most relevant to quantum computing. The general version of the CSD Theorem does not restrict the dimension of $U$ to be even, or even restrict the blocks into which $U$ is partitioned to be of equal size.
in Fig 2. The recursion connects these nodes so as to form a binary tree, as shown in Fig 3. In Fig 3 we start with an initial unitary matrix $U_{in}$ entering the root node, which we define as level 0. Without loss of generality, we can assume that the dimension of $U_{in}$ is $2^{N_B}$ for some $N_B \geq 1$. (If initially $U_{in}$’s dimension is not a power of 2, we replace it by a direct sum $U_{in} \oplus I_r$ whose dimension is a power of two.) We apply the CSD method to $U_{in}$. This yields for level 0 a D matrix $D^{(1)}_0$, two unitary matrices $L^{(1)}_0$ and $L^{(1)}_1$ on the left side and two unitary matrices $R^{(1)}_0$ and $R^{(1)}_1$ on the right side. Then we apply the CSD method to each of the 4 matrices $L^{(1)}_0$, $L^{(1)}_1$, $R^{(1)}_0$ and $R^{(1)}_1$ that were produced in the previous step. Then we apply the CSD method to each of the 16 $R$ and $L$ matrices that were produced in the previous step. And so on. The nodes of level $N_B$ don’t have $R, L$ arrows coming out of them since the $D$ matrices for those nodes are all $1 \times 1$.

Call a central matrix either (1) a single D matrix, or (2) a direct sum $D_1 \oplus D_2 \oplus \cdots \oplus D_r$ of D matrices, or (3) a diagonal unitary matrix. From Fig 3 it is clear that the initial matrix $U_{in}$ can be expressed as a product of central matrices, with each node of the tree providing one of the central matrices in the product. We can use this factorization of $U_{in}$ into central matrices to compile $U_{in}$, if we can find a method for decomposing any central matrix into a SEO. Tuc99 gives such a method.

Let’s say a few more words about central matrices. For simplicity, consider matrices of size $2^{N_B}$ with $N_B = 3$. Let $R_y(\phi) = e^{i\phi\sigma_Y}$. An $8 \times 8$ D matrix is of the form

$$D = \sum_{a,b \in \text{Bool}} R_y(\phi_{ab}) \otimes P_a \otimes P_b ,$$  

(49a)

for some $\phi_{ab} \in \mathbb{R}$. An $8 \times 8$ matrix which is a sum of two $D$ matrices is of the form

$$D_0 \oplus D_1 = \sum_{a,b \in \text{Bool}} P_a \otimes R_y(\phi'_{ab}) \otimes P_b ,$$  

(49b)

for some $\phi'_{ab} \in \mathbb{R}$. An $8 \times 8$ matrix which is a sum of four $D$ matrices is of the form

---

Figure 3: A CSD binary tree. It arises from the recursive application of the CSD.
\[ D_{00} \oplus D_{01} \oplus D_{10} \oplus D_{11} = \sum_{a,b \in \text{Bool}} P_a \otimes P_b \otimes R_y(\phi''_{ab}) , \]  
(49c)

for some \( \phi''_{ab} \in \mathbb{R} \). Thus, the \( D, D_0 \oplus D_1 \) and \( D_{00} \oplus D_{01} \oplus D_{10} \oplus D_{11} \) matrices given by Eqs.\( (49) \) only differ by a qubit permutation (apart from the fact that they have different angles). Note that \( D \) of Eq.\((49a)\) can also be written in the form

\[ D = \exp \left( i \sum_{a,b \in \text{Bool}} \phi_{a,b} \sigma_Y \otimes P_a \otimes P_b \right) \]  
(50a)

\[ = \exp \left( i \sum_{a,b \in \text{Bool}} \phi_{a,b} \sigma_Y(2)P_a(1)P_b(0) \right) . \]  
(50b)

For \( N_B \geq 1 \), Eq.\((50)\) generalizes to

\[ D = \exp \left( i \sigma_Y \otimes \sum_{\vec{b} \in \text{Bool}^{N_B-1}} \phi_{\vec{b}} P_{\vec{b}} \right) \]  
(51a)

\[ = \exp \left( i \sigma_Y(N_B-1) \sum_{\vec{b} \in \text{Bool}^{N_B-1}} \phi_{\vec{b}} P_{\vec{b}}(N_B-2, \ldots, 2, 1, 0) \right) . \]  
(51b)

In general, a central matrix acting on \( N_B \) qubits is a matrix of the form exhibited by \( D \) in Eq.\((51)\), or a qubit permutation thereof, or a diagonal unitary matrix.

In my papers that followed Tuc99, I’ve begun calling a \( D \) matrix of the form Eq.\((51)\) a “multiplexor” \( ^4 \). When I want to be more precise, I call it an \( R_y(2) \)-multiplexor with target qubit \( N_B - 1 \) and control qubits \( N_B - 2, \ldots, 2, 1, 0 \). The \( R_y(2) \) term refers to the fact that the set of operations acting on the target qubit are \( 2 \times 2 \) qubit rotations \( R_y(\phi) = e^{i\phi \sigma_Y} \) for some \( \phi \in \mathbb{R} \). More generally, one can speak of \( U(2) \)-multiplexors. Henceforth in this paper, I’ll continue using this multiplexor nomenclature, even though it’s not used in Tuc99.

Tuc99 gives identities for decomposing an arbitrary \( R_y(2) \)-multiplexor and a diagonal unitary matrix into a SEO with \( 2^N_B \) CNOTs. Fig.\( 4 \) shows an example of the SEO decomposition found in Tuc99 for an \( R_y(2) \)-multiplexor. In Fig.\( 4 \) 0,1,2,3 are the control qubits, and 4 is the target qubit. The empty square vertices represent \( R_y(2) \) gates. The symbol to the left of the equal sign, the one with the “half-moon” vertices, was invented by the authors of Ref.\( 18 \) to represent an \( R_y(2) \)-multiplexor.

The Tuc99 algorithm is implemented in a computer program called Qubiter, available at Ref.\( 19 \).

\( ^4 \)“multiplexor” means “multi-fold” in Latin. A special type of electronic device is also called a multiplexor or multiplexer.
Later papers\cite{20, 18, 21} have improved the Tuc99 algorithm. Instead of growing the CSD tree all the way down to where the leaf nodes contain diagonal unitary matrices, they stop one level earlier, when the leaf nodes contain $U(2)$-multiplexors. Then they apply a nice technique, due to Vidal and Dawson\cite{22}, for expressing arbitrary $U(2)$ matrices using just 3 CNOTs. They are able to express an arbitrary structureless $U(2^{NB})$ matrix using $\frac{1}{4}4^{NB} + O(2^{NB})$ CNOTs (and also some qubit rotations, of course). It’s easy to show\cite{23} that the number of CNOTs needed to express a structureless $U(2^{NB})$ matrix is greater or equal to $\frac{1}{4}(4^{NB} - 3NB - 1)$. Roughly speaking, in these algorithms, a unitary matrix is decomposed into $2^{NB} U(2)$-multiplexors, and each of these multiplexors is expressed as a SEO with $2^{NB}$ CNOTs.

Note that quantum compiling papers usually measure circuit complexity by counting the total number of CNOTs. Others count the total number of elementary gates. One could also count just the total number of control vertices in the circuit diagram. These 3 measures are linearly related so they have the same big O behavior when expressed as a function of the number of qubits. It is expected that the quantum computers of the future will be able to perform single-qubit operations much faster than CNOTs, so it makes sense to ignore the single-qubit operations and count only the CNOTs.

The Tuc99 algorithm and its variants expect a structureless unitary matrix as input. They do not assume any symmetries in the structure of the input matrix being decomposed. But the CSD does have some free degrees of freedom (for example, certain phase choices) which are chosen arbitrarily. The number of these degrees of freedom increases as the symmetry of the input matrix increases. If the input matrix does have a symmetric structure, it is sometimes possible to chose those free degrees of freedom in a way that reduces the length of the output SEO. For example, Refs.\cite{24, 25} show that the Coppersmith decomposition\cite{26} of the discrete Fourier transform matrix can be obtained using the Tuc99 algorithm and some judicious choices for the free degrees of freedom.
4 Some Identities for Exponentials of Partitioned Matrices

In this section, we present some identities satisfied by exp(i\(M\)), where \(M\) is a Hermitian matrix partitioned into four equal sized blocks.

Suppose \(M_{00}, M_{01}, M_{10}\) and \(M_{11}\) are square matrices of the same dimension, and

\[
M = \begin{bmatrix} M_{00} & M_{01} \\ M_{10} & M_{11} \end{bmatrix}.
\]  

(52)

We will say \(M\) is block diagonal if \(M_{10} = M_{01} = 0\) and block anti-diagonal if \(M_{00} = M_{11} = 0\). We are interested in the case were \(M\) is Hermitian.

If the \(M\) of Eq.(52) is block diagonal, then exp(i\(M\)) = exp(i\(M_{00}\)) \(\oplus\) exp(i\(M_{11}\)).

But suppose \(M\) is block anti-diagonal. What can we say about exp(i\(M\)) then?

**Lemma 1** Suppose \(F \in \mathbb{C}^{n \times n}\) and \(F = V\Delta U^\dagger\) is an SVD, so \(V, U\) are unitary matrices and \(\Delta\) is a non-negative diagonal matrix. Then

\[
\exp\left(i \begin{bmatrix} 0 & F^\dagger \\ F & 0 \end{bmatrix} \right) = \begin{bmatrix} U & 0 \\ 0 & V \end{bmatrix} \begin{bmatrix} \cos(\Delta) & i\sin(\Delta) \\ i\sin(\Delta) & \cos(\Delta) \end{bmatrix} \begin{bmatrix} U^\dagger & 0 \\ 0 & V^\dagger \end{bmatrix}.
\]  

(53)

**proof:** Let LHS and RHS stand for the left and right hand sides of Eq.(53). Then

\[
\text{LHS} = \exp\left(i \begin{bmatrix} U & 0 \\ 0 & V \end{bmatrix} \begin{bmatrix} 0 & \Delta \\ \Delta & 0 \end{bmatrix} \begin{bmatrix} U^\dagger & 0 \\ 0 & V^\dagger \end{bmatrix} \right) \quad (54a)
\]

\[
= \begin{bmatrix} U & 0 \\ 0 & V \end{bmatrix} \exp\left(i \begin{bmatrix} 0 & \Delta \\ \Delta & 0 \end{bmatrix} \right) \begin{bmatrix} U^\dagger & 0 \\ 0 & V^\dagger \end{bmatrix} \quad (54b)
\]

\[
= \text{RHS}.
\]  

(54c)

LHS can still be exponentiated without having to find the SVD of \(F\). See Appendix \(A\) if interested.

QED

Suppose the \(M\) of Eq.(52) only has \(M_{11} = 0\), so it isn’t fully block diagonal. What can we say about exp(i\(M\)) then?

**Lemma 2** Suppose \(A, B \in \mathbb{C}^{n \times n}\) and \(A\) is a Hermitian matrix. Then

\[
\exp\left(i \begin{bmatrix} A & B^\dagger \\ B & 0 \end{bmatrix} \right) = \exp\left(i \begin{bmatrix} A & 0 \\ 0 & 0 \end{bmatrix} \right) \mathcal{T} \left\{ \exp\left(i \int_0^1 \, dt \begin{bmatrix} 0 & h.c. \\ B e^{itA} & 0 \end{bmatrix} \right) \right\},
\]  

(55)

where \(\mathcal{T}\{\}\) indicates a time-ordered exponential.
first proof: Define the following three matrices:

\[
Q = \begin{bmatrix} A & B^\dagger \\ B & 0 \end{bmatrix}, \quad Q_A = \begin{bmatrix} A & 0 \\ 0 & 0 \end{bmatrix}, \quad Q_B = \begin{bmatrix} 0 & B^\dagger \\ B & 0 \end{bmatrix}.
\] (56)

For any \( t \in \mathbb{R} \), consider the \( t \)-dependent matrix

\[
x(t) = e^{-itQ_A}e^{itQ}.
\] (57)

The derivative of \( x(t) \) is

\[
x'(t) = e^{-itQ_A}(-iQ_A + iQ)e^{itQ} \quad \text{(58a)}
\]

\[
= (ie^{-itQ_A}Q_B e^{itQ_A})x(t). \quad \text{(58b)}
\]

Hadamard’s identity\[13\] tells us that

\[
e^{-itQ_A}Q_B e^{itQ_A} = e^{-it[Q_A,]Q_B}.
\] (59)

It is easy to check that for \( n = 1, 2, \ldots \),

\[
[Q_A,]^nQ_B = \begin{bmatrix} 0 & A^nB^\dagger \\ B(-A)^n & 0 \end{bmatrix}.
\] (60)

Therefore,

\[
ie^{-itQ_A}Q_B e^{itQ_A} = i \sum_{n=0}^{\infty} \frac{(-it)^n}{n!} \begin{bmatrix} 0 & A^nB^\dagger \\ B(-A)^n & 0 \end{bmatrix} \quad \text{(61a)}
\]

\[
= i \begin{bmatrix} 0 & e^{-itA}B^\dagger \\ Be^{itA} & 0 \end{bmatrix}. \quad \text{(61b)}
\]

Thus, \( x(t) \) defined by Eq.(57) satisfies the differential equation:

\[
x'(t) = i \begin{bmatrix} 0 & e^{-itA}B^\dagger \\ Be^{itA} & 0 \end{bmatrix} x(t). \quad \text{(62)}
\]

This differential equation for \( x(t) \) is of the form

\[
x'(t) = M(t)x(t),
\] (63)

where \( M(t) \) is a \( t \)-dependent matrix. If we express the derivative on the left hand side of Eq.\(\text{(63)}\) in terms of infinitesimals, we find

\[
x(t + dt) \approx [1 + M(t)dt]x(t) \approx e^{M(t)dt}x(t). \quad \text{(64)}
\]

Iterating Eq.\(\text{(64)}\) gives:
\[ x(t) = T \left\{ e^{\int_0^t dt'M(t')} \right\} x(0). \quad (65) \]

Eq. (65) is the solution of any differential equation of the form Eq. (63). We can use Eq. (65) to solve Eq. (62) with the initial condition \( x(0) = 1 \) (from Eq. (57)).

**second proof:** Here is an alternative proof that uses infinitesimals instead of differential equations. For any large integer \( N \), let \( \Delta t = \frac{1}{N} \) and \( t_k = k\Delta t \) for \( k \in \mathbb{Z}_{0,N} \).

Thus, \( t_0 = 0 \) and \( t_N = 1 \). Then

\[
\begin{align*}
    e^{-iQA}e^{iQ} & \approx e^{-iQA}(e^{i\Delta t Q_B}e^{i\Delta t Q_A})^N \\
    & \approx \prod_{j=N,...,2,1} \left\{ e^{-it_jQ_A}e^{i\Delta t Q_B}e^{it_jQ_A} \right\} \\
    & \approx \prod_{j=N,...,2,1} \exp(e^{-it_jQ_A}i\Delta t Q_B e^{it_jQ_A}) \\
    & \approx \prod_{j=N,...,2,1} \exp(i\Delta t \begin{bmatrix} 0 & e^{-it_j A}B^\dagger \\ B e^{it_j A} & 0 \end{bmatrix}) \\
    & = T \left\{ \exp(i \int_0^1 dt \begin{bmatrix} 0 & e^{-it A}B^\dagger \\ B e^{it A} & 0 \end{bmatrix}) \right\}. \quad (66e)
\end{align*}
\]

QED

In the previous Lemma, the magnitude of the matrices \( A \) and \( B \) was arbitrary. If we assume that \( A \) and \( B \) are \( \mathcal{O}(g) \), where \( g \) is small, then one can say more about the exponential of an \( M \) such that \( M_{11} = 0 \).

**Lemma 3** Suppose \( A, B \in \mathbb{C}^{n \times n} \), \( A \) is a Hermitian matrix, and \( A, B \) are both \( \mathcal{O}(g) \), where \( g \) is small. Then

\[
\exp(i \begin{bmatrix} A & B^\dagger \\ B & 0 \end{bmatrix}) = \exp(i \begin{bmatrix} A & 0 \\ 0 & 0 \end{bmatrix}) \exp\left( \begin{bmatrix} T_1 + T_2 & iB^\dagger \\ iB & -iBAB^\dagger \end{bmatrix} \right) + \mathcal{O}(g^5), \quad (67)
\]

where

\[
T_1 = \frac{i}{12} (B^\dagger BA + AB^\dagger B), \quad (68)
\]

\[
T_2 = \frac{1}{24} (-B^\dagger BA^2 + A^2 B^\dagger B), \quad (69)
\]

\[
\overline{B} = B \operatorname{sinc}(\frac{A}{2})e^{i\frac{\pi}{4}}. \quad (70)
\]

Also, to lower order,
\[
\exp(i \begin{bmatrix} A & B^\dagger \\ B & 0 \end{bmatrix}) = \exp(i \begin{bmatrix} A & 0 \\ 0 & 0 \end{bmatrix}) \exp(i \begin{bmatrix} 0 & B^\dagger \\ B & 0 \end{bmatrix}) + \mathcal{O}(g^3). \tag{71}
\]

**proof:** (The arXiv source code for this paper includes an Octave/Matlab subroutine called qtree_cbh.m that checks this Lemma.) If \( x \) and \( y \) are both of order \( g \) and \( g \) is small, then the Campbell-Baker-Hausdorff Expansion is \([13, 14]\)

\[
e^x e^y = \exp \left\{ x + y + \frac{1}{2} [x, y] + \frac{1}{12} ([x, .]^2 y + [y, .]^2 x) + \frac{1}{24} [y, [x, [y, x]]] + \mathcal{O}(g^5) \right\}. \tag{72}
\]

If \( Q_A \) and \( Q_B \) are defined by Eq.(56), then this Lemma follows from Eq.(72) with \( x = -iQ_A, \ y = i(Q_A + Q_B) \).

The definition of \( \overline{B} \) was inspired from Lemma [2] and the observation that

\[
\int_0^1 dt \ e^{itA} = \frac{e^{iA} - 1}{iA} = \text{sinc}(\frac{A}{2}) e^{i\frac{A}{2}}, \tag{74}
\]

where \( \text{sinc}(x) = \sin(x)/x \) as usual.

QED

5 General Strategy for Compiling Any Quantum Walk

In this section, we describe our general strategy for compiling the FGG07 algorithm (or, more precisely, the FGG07-inspired algorithm with discrete queries that was first proposed in Cle07). The compilation strategy described in this section is also useful for compiling other quantum walks.

The standard definition of the evolution operator in Quantum Mechanics is \( U = e^{-itH} \), where \( t \) is time and \( H \) is a Hamiltonian. Throughout this paper, we will set \( t = -1 \) so \( U = e^{iH} \). If \( H \) is proportional to a coupling constant \( g \), reference to time can be restored easily by replacing the symbol \( g \) by \(-tg\), and the symbol \( H \) by \(-tH\).

Fig.5(a) shows an example of a “loop” graph. It has 8 nodes (nodes=states), labelled 0, 1, …, 7. We wish to consider the following Hamiltonian for transitions along the edges of this loop graph:
Figure 5: (a) loop graph (b) tree graph (c) tree and loop graphs glued together

\[ H_{lp} = (g) \begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
0 & & & & & & & 1 \\
1 & 1 & & & & & & \\
2 & 1 & 1 & & & & & \\
3 & 1 & 1 & 1 & & & & \\
4 & & 1 & 1 & & & & \\
5 & & & 1 & 1 & & & \\
6 & & & & & 1 & 1 & \\
7 & 1 & & & & & & \\
\end{array} \]

where \( g \in \mathbb{R} \).

A binary tree with \( \Lambda + 1 \) levels has \( 1 + 2 + 2^2 + \ldots + 2^{\Lambda} = 2^{\Lambda+1} - 1 \) nodes. To reach \( 2^{\Lambda+1} \) nodes, we add an extra “dead” or “dud” node, labelled with the letter “d”. This d node is not connected to any other node in the graph. If we include this dud node, then the number of leaves \( N_{\text{lv}} \) is exactly half the number of nodes: \( N_{\text{lv}} = \frac{1}{2}2^{\Lambda+1} = 2^\Lambda \).

Fig.5(b) shows an example of a binary-tree graph with \( \Lambda = 2 \). It has 8 nodes, labelled \( d, 1, 2, \ldots, 7 \). We wish to consider the following Hamiltonian for transitions along the edges of this binary-tree graph:
where \( g_{p,q} \in \mathbb{R} \) for all \( p,q \).

Fig. 5(c) shows an example of the graph used in FGG07. It contains a binary tree attached to a loop which serves as a “runway” for a wave packet. For the algorithm of FGG07 to work, the runway must be much longer than \( \sqrt{N_{\text{lv}}\ell_s} \) as \( N_{\text{lv}} \to \infty \). In Fig. 5(c), the tree has 8 nodes (including the dud one), and the runway has 8 nodes. Let’s call the tree nodes \( tr_j \), where \( j \in \mathbb{Z}_{0,7} \), and the runway nodes \( lp_j \), where \( j \in \mathbb{Z}_{0,7} \). We wish to consider the following Hamiltonian for transitions along the edges of this tree and runway graph:

\[
H = \begin{bmatrix}
H_{tr} & h^\dagger_{gl} & h_{gl} & h_{in} \\
\sqrt{h}_{gl} & H_{lp} & \sqrt{h^\dagger}_{gl} & \sqrt{h}_{in} \\
\sqrt{h}_{in} & \sqrt{h^\dagger}_{in} & H_{tr} & \sqrt{h}_{gl} \\
\sqrt{h_{in}} & \sqrt{h^\dagger_{in}} & \sqrt{h_{gl}} & H_{lp}
\end{bmatrix}.
\]  \tag{77}

If \( g \) is the coupling constant that appears in the loop Hamiltonian \( H_{lp} \), then we will assume, as FGG07 does, that the coupling constants \( \{ g_{p,q} \}_{q_p,q} \) that appear in the tree Hamiltonian \( H_{tr} \) are all equal to \( g \). Besides \( H_{tr} \) and \( H_{lp} \), which have already been discussed, the Hamiltonian of Eq. (77) includes blocks with \( h_{gl} \), \( h_{in} \), and their Hermitian conjugates. The tree-loop “glue” Hamiltonian \( H_{gl} \) is defined by

\[
H_{gl} = h_{gl} + h^\dagger_{gl} = g(\langle lp_0 | tr_1 \rangle + \text{h.c.}).
\]  \tag{78}

It corresponds to the graph edge that connects the tree and the runway. The input (oracle) Hamiltonian \( H_{in} \) is defined by

\[
H_{in} = h_{in} + h^\dagger_{in} = g \sum_k x_k (\langle tr_k | in_k \rangle + \text{h.c.}),
\]  \tag{79}

where \( k \) runs over all tree leaves, and \( x_k \in \text{Bool} \) are the NAND formula inputs.

\[5\] FGG07 “requests” a runway of length \( N_{\text{lv}}\ell_s \). However, only a stretch of length \( c\sqrt{N_{\text{lv}}\ell_s} \), for some large constant \( c \), is ever used (traversed) by the narrow wavepacket during the duration of the experiment. Chi07 and Amb07 request a tail of length \( c\sqrt{N_{\text{lv}}\ell_s} \).
The compilation strategy of this paper is to split the full Hamiltonian of Eq.(77) into two parts, which we call the **bulk Hamiltonian** $H_{\text{bulk}}$ and the **boundary corrections Hamiltonian** $H_{\text{corr}}$:

$$H = H_{\text{bulk}} + H_{\text{corr}}.$$  \hspace{1cm} (80)

We will set

$$H_{\text{bulk}} = H_{\text{lp}} + H_{\text{tr}} \quad , \quad H_{\text{corr}} = H_{\text{gl}} + H_{\text{in}}.$$  \hspace{1cm} (81)

Note that $[H_{\text{lp}}, H_{\text{tr}}] = 0$ and $[H_{\text{gl}}, H_{\text{in}}] = 0$. In subsequent sections we will show how to compile the evolution operators $\exp(iH_{\text{lp}})$ and $\exp(iH_{\text{tr}})$, and therefore $\exp(iH_{\text{bulk}})$. We will also show how to compile $\exp(iH_{\text{gl}})$ and $\exp(iH_{\text{in}})$, and therefore $\exp(iH_{\text{corr}})$. Exact compilation of $\exp(iH_{\text{bulk}} + iH_{\text{corr}})$, although possible via an algorithm such as that of Tuc99, is numerically very laborious. Our strategy is to avoid such numerical calculation by combining the compilations of $\exp(iH_{\text{bulk}})$ and $\exp(iH_{\text{corr}})$ via the Trotterized Suzuki approximation.

Actually, FGG07 assumes a runway that is a straight line, but replacing a straight line by a large enough closed loop does not change the algorithm significantly. Purists can always convert the loop Hamiltonian into a line Hamiltonian by adding an extra loop-cutting Hamiltonian

$$H_{\text{lp-cut}} = -g(|lp_3\rangle \langle lp_4| + \text{h.c.})$$  \hspace{1cm} (82)

to the corrections Hamiltonian. It is also possible to convert a loop into a “tail” like the one used in Chi07 and Amb07, by adding a loop-cutting Hamiltonian that cuts the loop at the edge that connects nodes $lp_0$ and $lp_7$.

### 6 Compiling Loop Graphs

In this section, we will show how to compile (exactly) the evolution operator $e^{iH_{\text{lp}}}$ for a loop graph.

Eq.(75) is an example of a loop Hamiltonian $H_{\text{lp}}$. Since $H_{\text{lp}}$ is a circulant matrix, $e^{iH_{\text{lp}}} = U e^{iD} U^\dagger$, where $U$ is a discrete Fourier transform matrix and $D$ is a diagonal real matrix. The $U$ and $U^\dagger$ can be immediately compiled via the Copper-Smith decomposition\cite{26}, using $O(N_B^2)$ CNOTs, where $N_S = 2^{N_B}$ is the number of states in the loop. $e^{-iD}$ is a diagonal unitary matrix. Tuc99 shows how to compile an $N_S \times N_S$ diagonal unitary matrix using $O(N_S)$ CNOTs.

Suppose $\theta_b \in \mathbb{R}$ for all $b \in \text{Bool}^{N_B}$. Suppose $\vec{\beta}$ are $N_B$ distinct qubit positions
and $\alpha$ is the position of an additional ancilla qubit. Note that

$$\exp \left( i \sum_{\beta} \theta_{\beta} P_{\beta}(\vec{\beta}) \right) = \exp \left( i \sigma_Z(\alpha) \sum_{\beta} \theta_{\beta} P_{\beta}(\vec{\beta}) \right) |0\rangle_\alpha$$  \hspace{1cm} (83a)

$$= e^{-i \pi \sigma_X(\alpha)} \exp \left( i \sigma_Y(\alpha) \sum_{\beta} \theta_{\beta} P_{\beta}(\vec{\beta}) \right) e^{i \pi \sigma_X(\alpha)} |0\rangle_\alpha$$ \hspace{1cm} (83b)

We used $\sigma_Z|0\rangle = |0\rangle$ and $e^{-i \pi \sigma_X} \sigma_Y e^{i \pi \sigma_X} = \sigma_Z$. Thus, if one can compile any $R_y(2)$-multiplexor with $N_B$ controls, then one can compile any $2^{N_B} \times 2^{N_B}$ diagonal unitary matrix.

7 Compiling Glue and Cuts

In this section, we will show how to compile the evolution operators $e^{iH_{\text{cut}}}$ for a loop cut and $e^{iH_{\text{gl}}}$ for the tree-loop glue.

Consider the loop cut first. For example, suppose we want to cut the loop graph of Fig.5(a) at the edge connecting the states $|0\rangle$ and $|7\rangle$.

Define

$$U = [\sigma_X(2)\sigma_X(1)]^{n(0)}.$$  \hspace{1cm} (84)

Then

$$|7\rangle \langle 0| + \text{h.c.} = |111\rangle \langle 000| + \text{h.c.} \hspace{1cm} (85a)$$

$$= U(|001\rangle \langle 000| + \text{h.c.}) U^\dagger \hspace{1cm} (85b)$$

$$= U \sigma_X(0)^{\vec{1}(2)\vec{1}(1)} U^\dagger.$$  \hspace{1cm} (85c)

To get Eq.(85c), we used the fact that $\sigma_X(\beta) = (|0\rangle \langle 1| + \text{h.c.})_\beta$ for any qubit position $\beta$. Finally, note that

$$e^{iH_{\text{cut}}} = e^{-i g |7\rangle \langle 0| + \text{h.c.}} = U [e^{-i g \sigma_X(0)^{\vec{1}(2)\vec{1}(1)}}] U^\dagger.$$  \hspace{1cm} (86)

In general, suppose we want to cut the edge that connects states $|j\rangle$ and $|k\rangle$, where $j$ and $k$ are the decimal names of the states. Let $\vec{j} = \text{bin}(j)$ and let $j_\beta$ be the component of $\vec{j}$ at bit position $\beta$. Define $\vec{k}$ and $k_\beta$ analogously. $j \neq k$ so $\vec{j}$ and $\vec{k}$ must differ at one bit position, at least. Let $\beta_o$ be one such bit position. Assume $j_{\beta_o} = 1$ and $k_{\beta_o} = 0$. Define

$$U = \prod_{\beta: j_\beta \neq k_\beta, \beta \neq \beta_o} \sigma_X(\beta)^{n(\beta_o)}.$$  \hspace{1cm} (87)

Then
\[ |j > < k| + h.c. = U \sigma_X(\beta_o) \prod_{\beta, \beta \neq \beta_o} P_{k, \beta(\beta)} U^\dagger. \] (88)

Therefore,
\[ e^{iH_{cut}} = e^{-ig(j|k| + h.c.)} U [e^{-ig\sigma_X(\beta_o)} \prod_{\beta, \beta \neq \beta_o} P_{k, \beta(\beta)} U^\dagger. \] (89)

Compiling \( e^{iH_{gl}} \) is identical to compiling \( e^{iH_{cut}} \). The only difference is that the coupling constant \( g \) is replaced by \( -g \).

## 8 Compiling Line Graphs

In this section, we will show how to compile (approximately) the evolution operator \( e^{iH_{line}} \) of a line graph (open loop). Previously, we showed how to compile the evolution operators \( e^{iH_{lp}} \) and \( e^{iH_{cut}} \) for a loop graph and a cut. A possible compilation of \( e^{iH_{line}} \) can be achieved by combining \( e^{iH_{lp}} \) and \( e^{iH_{cut}} \) via Trotterized Suzuki. Alternatively, one can compile \( e^{iH_{line}} \) directly, as will be shown in this section.

Eq. (75) is an example of a line graph Hamiltonian \( H_{line} \), provided we set to zero the entries \((0, 7)\) and \((7, 0)\). In Eq. (75), we’ve label states by the decimal numbers 0 to 7, and we have assumed that transitions between these states are such that their label \( x \in \mathbb{Z}_{0, 7} \) can only vary by \( \Delta x = \pm 1 \). As in spectroscopy, let’s call this constraint on \( \Delta x \) a selection rule. Although compiling \( e^{iH_{line}} \) with an \( H_{line} \) that satisfies \( \Delta x = \pm 1 \) is possible, a cleaner, simpler compilation\(^8\) can be achieved if \( H_{line} \) satisfies a different selection rule; namely, the constraint that states can only change to other states iff the initial and final states are adjacent in a Gray ordering.

In a Gray ordering, states are labelled by a binary number, and adjacent states have labels that differ only at one bit position.

\[ \begin{array}{cccccccc}
000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \\
000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \\
010 & 011 & 100 & 101 & 110 & 111 & 100 & 101 \\
011 & 110 & 111 & 100 & 101 & 110 & 111 & 100 \\
\end{array} \]

\[ \text{DEC GRAY} \]

Figure 6: Line graph for eight states. States are connected iff their binary representations differ only at one bit position; i.e., connected states are adjacent in a Gray ordering.

Fig. 6 shows eight states, ordered linearly in a decimal ordering from left to right. Edges connecting the states indicate possible transitions between the states. These transitions can only occur between states that are adjacent in a Gray ordering.
The Hamiltonian $H_{\text{line}}$ that describes transitions along the edges of the graph of Fig. 6 is the following. (Note that the states along the rows and columns of $H_{\text{line}}$ are ordered in a decimal ordering.)

\[
H_{\text{line}} = (g)
\begin{array}{cccccccc}
000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \\
000 & 0 & 1 & & & & & \\
001 & 1 & 0 & 1 & & & & \\
010 & 0 & 1 & 1 & & & & \\
011 & 1 & 1 & 0 & & & & \\
100 & & & 0 & 1 & & & \\
101 & & & 1 & 0 & 1 & & \\
110 & & & & 1 & 0 & 0 & \\
111 & & & & & 1 & 1 & 0
\end{array}
\] (90)

\[
= g\begin{bmatrix}
\sigma_X & P_1 \\
P_1 & \sigma_X & P_0 \\
P_0 & P_0 & \sigma_X
\end{bmatrix}
\] (91)

\[
= g\begin{bmatrix}
\sigma_X & \sigma_X \\
\sigma_X & \sigma_X \\
\sigma_X & \sigma_X
\end{bmatrix}
+ \begin{bmatrix} \rho_1 & \rho_1 \\
\rho_1 & \rho_0 \end{bmatrix}
+ \begin{bmatrix} \rho_0 \\
\rho_0 \end{bmatrix}
\] (92)

\[
= g\{I \otimes I \otimes \sigma_X + I \otimes \sigma_X \otimes P_1 + \sigma_X \otimes P_1 \otimes P_0 \}
\] (93)

\[
= g\{\sigma_X(0) + \sigma_X(1)n(0) + \sigma_X(2)n(1)\overline{n}(0)\}.
\] (94)

where $g \in \mathbb{R}$. Define

\[
A = g\sigma_X(0)
\]
\[
B_1 = g\sigma_X(1)n(0)
\]
\[
B_2 = g\sigma_X(2)n(1)\overline{n}(0)
\] (95)

Note that the $B_j$ commute because they live in different states (see Fig. 6). Thus,

\[
e^{iH_{\text{line}}} = e^{i(A+B_1+B_2)} = e^{iA}e^{iB_1}e^{iB_2} + \mathcal{O}(g^2)
\] (96)

\[
e^{iH_{\text{line}}} = e^{iB_2} + \mathcal{O}(g^2)
\] (97)

if we use a first-order Lie approximant, and
\[
e^{iH_{line}} = e^{i(A+B_1+B_2)} = e^{i\frac{A}{2}}e^{iB_1}e^{iB_2}e^{i\frac{A}{2}} + \mathcal{O}(g^3) \tag{98}
\]

\[
e = R_x \downarrow \downarrow \downarrow \downarrow R_x + \mathcal{O}(g^3) \tag{99}
\]

if we use a second-order Suzuki approximant.

With four bits, one gets

\[
H_{line} = g
\]

\[
\begin{array}{c|c|c|c|c|c|c}
\sigma_X & P_1 & P_1 & \sigma_X & P_0 & P_0 & P_0 \\
P_1 & \sigma_X & P_1 & \sigma_X & P_0 & P_0 & P_0 \\
P_0 & P_1 & \sigma_X & P_1 & \sigma_X & P_0 & P_0 \\
P_0 & P_1 & \sigma_X & P_0 & P_0 & P_0 & P_0 \\
\end{array}
\]

\[
= g\{\sigma_X(0) + \sigma_X(1)n(0) + \sigma_X(2)n(1)\overline{n}(0) + \sigma_X(3)n(2)\overline{n}(1)\overline{n}(0)\} . \tag{100}
\]

where \(g \in \mathbb{R}\). Define

\[
A = g\sigma_X(0) \\
B_1 = g\sigma_X(1)n(0) \\
B_2 = g\sigma_X(2)n(1)\overline{n}(0) \\
B_3 = g\sigma_X(3)n(2)\overline{n}(1)\overline{n}(0)
\] \tag{102}

Then

\[
e^{iH_{line}} = e^{i(A+B_1+B_2+B_3)} = e^{iA}e^{iB_1}e^{iB_2}e^{iB_3} + \mathcal{O}(g^2) \tag{103}
\]

\[
e = R_x \downarrow \downarrow \downarrow \downarrow R_x + \mathcal{O}(g^2) \tag{104}
\]

if we use a first-order Lie approximant, and similarly if we use a second-order Suzuki approximant.

Generalization to arbitrary number of bits is obvious.
9 Generalities about Compiling Tree Graphs

In this section, we will introduce several key ideas that are useful in compiling the evolution operator \( \exp(iH_{tr}) \) for a general binary tree graph.

For \( \lambda = 1, 2, 3, \ldots \), define the \( 2^\lambda \times 2^\lambda \) matrix \( A_{2^\lambda} \) to be, up to a constant factor, the Hamiltonian \( H_{tr} \) for a binary tree with \( 2^\lambda \) nodes. By studying Eq.(76), one realizes that the family of matrices \( \{A_{2^\lambda}\}_\forall \) can be specified in terms of a family of matrices \( \{B_{2^\lambda}\}_\forall \) as follows. Let \( A_2 = 0 \), and

\[
A_{2^{\lambda+1}} = \begin{bmatrix} A_{2^\lambda} & B_{2^\lambda} \dagger \\ B_{2^\lambda} & 0 \end{bmatrix},
\]

for \( \lambda = 1, 2, 3, \ldots \). (The subscripts of \( A_{2^\lambda} \) and \( B_{2^\lambda} \) indicate the dimension of these matrices.) For example, according to Eq.(105), \( A_{16} \) has the form:

\[
A_{16} = \begin{bmatrix}
\begin{array}{c|c|c|c}
0_2 & B_{2}^\dagger & B_{4}^\dagger & B_{8}^\dagger \\
B_{2} & 0_2 & B_{4} & 0_4 \\
B_{4} & 0_4 & B_{8} & 0_8 \\
B_{8} & 0_8 & & \\
\end{array}
\end{bmatrix}.
\]

9.1 CSD-ready Evolutions

By applying Eq.(71) repeatedly, we can factor \( e^{iA_{16}} \) with \( A_{16} \) given by Eq.(106) as follows:

\[
e^{iA_{16}} = \Gamma_1 \Gamma_2 \Gamma_3 + \mathcal{O}(g^3),
\]

where

\[
\Gamma_3 = \exp(i \begin{bmatrix} 0 & B_{8}^\dagger \\ B_{8} & 0 \end{bmatrix}),
\]

\[
\Gamma_2 = \exp(i \begin{bmatrix} 0 & B_{4}^\dagger \\ B_{4} & 0 \end{bmatrix}),
\]

and

\[
\Gamma_1 = \exp(i \begin{bmatrix} 0 & B_{2}^\dagger \\ B_{2} & 0 \end{bmatrix}).
\]
$B_{2\lambda}$ is defined for $\lambda \in \mathbb{Z}_{1,3}$ by

$$B_{2\lambda} = B_{2\lambda} \operatorname{sinc}(\frac{A_{2\lambda}}{2})e^{i\frac{A_{2\lambda}}{2}}.$$  \hspace{1cm} (109)

The above example generalizes as follows. For $\Lambda = 1, 2, 3, \ldots$, and $\lambda \in \mathbb{Z}_{1,\Lambda}$, we have

$$e^{iA_{2\Lambda+1}} = \Gamma_1 \Gamma_2 \ldots \Gamma_\Lambda + \mathcal{O}(g^3),$$  \hspace{1cm} (110)

$$\Gamma_\lambda = \exp(i \begin{bmatrix} 0 & B_{2\lambda}^d \cr B_{2\lambda} & 0 \cr \end{bmatrix}) \Gamma_0 \Gamma_{2\lambda-2\Lambda+1},$$  \hspace{1cm} (111)

and

$$B_{2\lambda} = B_{2\lambda} \operatorname{sinc}(\frac{A_{2\lambda}}{2})e^{i\frac{A_{2\lambda}}{2}}.$$  \hspace{1cm} (112)

Since these $\Gamma_\lambda$ are ready for an application of the CSD, we will henceforth refer to them as \textit{CSD-ready evolutions (or factors)} of $\exp(iA_{2\Lambda+1})$, and to Eq. (110) as a factorization of $\exp(iA_{2\Lambda+1})$ into CSD-ready factors.

### 9.2 SVD of $B_{2\lambda}$

Next we will calculate an SVD for each of the matrices $B_{2\lambda}$ that appear in Eq. (105). These SVD’s will be useful to us in future sections.

We will use $E_4$ to denote the $4 \times 4$ matrix that exchanges (swaps) 2 qubits:

$$E_4 = \begin{bmatrix} 00 & 01 & 10 & 11 \\ 00 & 1 & & \\ 01 & & 1 & \\ 10 & & & 1 \\ 11 & & & \\ \end{bmatrix}.$$  \hspace{1cm} (113)

Given $a_j, b_j \in \mathbb{R}$ for some index $j$, define

$$\rho_j = \sqrt{a_j^2 + b_j^2}, \quad U_j = \frac{1}{\rho_j} \begin{bmatrix} b_j & a_j \\ -a_j & b_j \\ \end{bmatrix}.$$  \hspace{1cm} (114)

The matrix $B_2$ that appears in Eq. (105) is of the form

$$B_2 = \begin{bmatrix} 0 & a \\ 0 & b \\ \end{bmatrix}.$$  \hspace{1cm} (115)

\hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{1cm} \hspace{
These values of $a$ and $b$ define a matrix $U$ and a scalar $\rho$ via Eq.(114) (with the $j$ subscript absent). The SVD of $B_2$ is one-sided and can be found in closed form. It’s given by:

$$B_2 = F_2 \begin{bmatrix} 0 & 0 \\ 0 & \Delta_1 \end{bmatrix} ,$$

(116)

where

$$F_2 = U ,$$

(117a)

and

$$\Delta_1 = \rho .$$

(117b)

When expressed in operator rather than matrix notation, Eqs.(117) become

$$F_2(0) = U(0) ,$$

(118a)

and

$$\Delta_1 = \rho .$$

(118b)

The matrix $B_4$ that appears in Eq.(105) is of the form

$$B_4 = \begin{bmatrix} a_0 & 0 \\ b_0 & 0 \\ 0 & a_1 \\ 0 & b_1 \end{bmatrix} ,$$

(119)

For $j = 0, 1$, these values of $a_j$ and $b_j$ define a matrix $U_j$ and a scalar $\rho_j$ via Eq.(114). The SVD of $B_4$ is one-sided and can be found in closed form. It’s given by:

$$B_4 = \begin{bmatrix} U_0 \\ U_1 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ \rho_0 & 0 \\ 0 & 0 \\ 0 & \rho_1 \end{bmatrix} ,$$

(120a)

$$= F_4 \begin{bmatrix} 0_2 \\ \Delta_2 \end{bmatrix} ,$$

(120b)

where

$$F_4 = (U_0 \oplus U_1)E_4 ,$$

(121a)

and

$$\Delta_2 = diag(\rho_0, \rho_1) .$$

(121b)
When expressed in operator rather than matrix notation, Eqs. (121) become

\[ F_4(1, 0) = U_0^{n(1)}(0)U_1^{n(1)}(0)E_4(1, 0) , \] (122a)

and

\[ \Delta_2(0) = \sum_{b \in \text{Bool}} \rho_b P_b(0) . \] (122b)

The matrix \( B_8 \) that appears in Eq. (105) is of the form

\[
B_8 = \begin{bmatrix}
| & | & | & | & | & | & | & | & | \\
| & | & | & | & | & | & | & | \\
| & | & | & | & | & | & | & | \\
| & | & | & | & | & | & | & | \\
| & | & | & | & | & | & | & | \\
| & | & | & | & | & | & | & | \\
| & | & | & | & | & | & | & | \\
| & | & | & | & | & | & | & | \\
| & | & | & | & | & | & | & | \\
| & | & | & | & | & | & | & | \\
\end{bmatrix} . \] (123)

For \( j \in \{00, 01, 10, 11\} \), these values of \( a_j \) and \( b_j \) define a matrix \( U_j \) and a scalar \( \rho_j \) via Eq. (114). The SVD of \( B_8 \) is one-sided and can be found in closed form. It’s given by:

\[
B_8 = \begin{bmatrix}
U_{00} & & & & \\
& U_{01} & & & \\
& & U_{10} & & \\
& & & U_{11} & \\
\end{bmatrix} \begin{bmatrix}
| & | & | & | & | & | & | & | & | \\
| & | & | & | & | & | & | & | \\
| & | & | & | & | & | & | & | \\
| & | & | & | & | & | & | & | \\
| & | & | & | & | & | & | & | \\
| & | & | & | & | & | & | & | \\
| & | & | & | & | & | & | & | \\
| & | & | & | & | & | & | & | \\
| & | & | & | & | & | & | & | \\
| & | & | & | & | & | & | & | \\
\end{bmatrix} \] (124a)

\[
= F_8 \begin{bmatrix}
0_4 & \\
& \Delta_4
\end{bmatrix} , \] (124b)

where

\[
F_8 = (U_{00} \oplus U_{01} \oplus U_{10} \oplus U_{11})(I \otimes E_4)(E_4 \otimes I) , \] (125a)

and

\[
\Delta_4 = \text{diag}(\rho_{00}, \rho_{01}, \rho_{10}, \rho_{11}) . \] (125b)
When expressed in operator rather than matrix notation, Eqs. (125) become

\begin{align}
F_8(2,1,0) &= U_{00}(0) m^{(2)(1)} U_{01}(0) m^{(2)} n^{(1)} U_{10}(0) n^{(2)} m^{(1)} U_{11}(0) n^{(2)} n^{(1)} E_4(1,0) E_4(2,1),
\end{align}

and

\begin{align}
\Delta_4(1,0) &= \sum_{\vec{b} \in \text{Bool}^2} \rho_{\vec{b}} P_{\vec{b}}(1,0).
\end{align}

The above example generalizes as follows. For \( \lambda = 1, 2, 3, \ldots \),

\begin{align}
F_{2\lambda}(\lambda-1, \ldots, 2, 1, 0) &= \prod_{\vec{b} = \text{Bool}^{\lambda-1}} \{ U_{\vec{b}}^{P_{\vec{b}}(\lambda-1, \ldots, 2, 1)}(0) \} E_4(0,1) E_4(1,2) \ldots E_4(\lambda-2, \lambda-1),
\end{align}

and

\begin{align}
\Delta_{2\lambda-1}(\lambda - 2, \ldots, 2, 1, 0) &= \sum_{\vec{b} \in \text{Bool}^{\lambda-1}} \rho_{\vec{b}} P_{\vec{b}}(\lambda - 2, \ldots, 2, 1, 0).
\end{align}

In this paper, we are interested mainly in the case of balanced binary NAND trees. For such trees, \( \rho_{\vec{b}} \) and \( U_{\vec{b}} \) are both independent of \( \vec{b} \). In particular,

\begin{align}
U_{\vec{b}} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} = H \sigma_X = e^{i \frac{\pi}{4} \sigma_y}
\end{align}

for all \( \vec{b} \). Therefore, for balanced binary trees, \( \Delta_{2\lambda} \) is proportional to the identity matrix and

\begin{align}
F_{2\lambda}(\lambda - 1, \ldots, 2, 1, 0) = (H \sigma_X)(0) E_4(0,1) E_4(1,2) \ldots E_4(\lambda-2, \lambda-1),
\end{align}

for all \( \lambda = 1, 2, 3 \ldots \).

## 10 Compiling Balanced Binary NAND Trees

In Section 9, we found an SVD for each matrix \( B_{2\lambda} \). Half of its singular values were zero and the other half were stored in a diagonal matrix called \( \Delta_{2\lambda-1} \). In this section, which contains several subsections, we will give an approximate compilation of \( \exp(i H_{tr}) \) for a special type of binary tree that includes balanced binary NAND trees. The special type of binary trees that we will consider in this section satisfies

\begin{align}
\Delta_{2\lambda} = d_{\lambda} I_{2\lambda},
\end{align}

for all possible \( \lambda \), where \( d_{\lambda} \in \mathbb{R} \). In other words, the \( \Delta_{2\lambda} \) of \( H_{tr} \) are all proportional to the identity matrix. Balance binary NAND trees satisfy Eq. (131) because their coupling constants \( g_{p,q} \) are the same for all \( p, q \).
10.1 Diagonalizing the Tree Levels

For $\lambda = 1, 2, 3, \ldots$, define

$$B_{2\lambda} = \begin{bmatrix} 0_{2^{\lambda-1}} & \Delta_{2^{\lambda-1}} \end{bmatrix}.$$  \hspace{1cm} (132)

A family of matrices $\{A_{2\lambda}\}_{\forall\lambda}$ can be defined in terms of the family of matrices $\{B_{2\lambda}\}_{\forall\lambda}$ as follows. Let $A_2 = 0$, and

$$A_{2\lambda+1} = \begin{bmatrix} A_{2\lambda} & B_{2\lambda}^\dagger \\ B_{2\lambda} & 0 \end{bmatrix}$$  \hspace{1cm} (133)

for $\lambda = 1, 2, 3, \ldots$. The subscripts of $A_{2\lambda}$ and $B_{2\lambda}$ indicate the dimension of these matrices.

Next, we will explain how the matrices $A_{2\lambda}$ are related to their namesakes $A_{2\lambda}$. For illustrative purposes, consider $A_{16}$ and $A_{16}$ first. We claim that if we define

$$\mathcal{F}_{16} = \text{diag}(I_8, F_8)\text{diag}(I_4, F_4, I_4, F_4)\text{diag}(I_2, F_2, I_2, F_2, I_2, F_2)$$
$$= (I_8 \oplus F_8)(I_4 \oplus F_4)^{\oplus 2}(I_2 \oplus F_2)^{\oplus 4},$$  \hspace{1cm} (134)

then

$$A_{16} = \mathcal{F}_{16}A_{16}\mathcal{F}_{16}^\dagger.$$  \hspace{1cm} (135)
This claim follows from the following observations:

\[ A_{16} = \begin{bmatrix}
  0_2 & B_2^\dagger & B_4^\dagger & B_8^\dagger \\
  B_2 & 0_2 & & \\
  B_4 & 0_4 & & \\
  B_8 & 0_8 & & \\
\end{bmatrix} \]

(136a)

\[ = \begin{bmatrix}
  I_4 & F_4^\dagger & I_4 & F_4^\dagger \\
  I_2 & F_2^\dagger & I_2 & F_2^\dagger & I_2 & F_2^\dagger \\
\end{bmatrix} \]

(136b)

\[ = \begin{bmatrix}
  I_4 & I_2 & F_4 & I_2 \\
  F_2 & F_2 & F_2 & F_2 \\
\end{bmatrix} \]

(136c)

\[ = \mathcal{F}_{16} \begin{bmatrix}
  0_2 & B_2^\dagger & B_4^\dagger & B_8^\dagger \\
  B_2 & 0_2 & & \\
  B_4 & 0_4 & & \\
  B_8 & 0_8 & & \\
\end{bmatrix} \mathcal{F}_{16}^\dagger . \]

(136d)

Going from Eq. (136a) to Eq. (136b) is the crucial step that requires assumption Eq. (131). Eq. (136b) is a non-conventional way of expressing the product of 3 matrices: the matrix that is totally enclosed by double lines is being pre-multiplied by matrix \( \mathcal{F}_{16} \) and post-multiplied by matrix \( \mathcal{F}_{16}^\dagger \). \( \mathcal{F}_{16} \) is spread out over the left margin and \( \mathcal{F}_{16}^\dagger \) over the top margin of the matrix enclosed by the double-line border. Writing Eq. (136b) in this nonconventional way makes it clear why assumption Eq. (131) is needed. For example, because \( \Delta_4 \) acts like a scalar, the matrix \( F_2^\dagger \) in the last column of the top margin commutes with \( \Delta_4 \) and can cancel the \( F_2 \) in the fourth row of the left margin.
Note that $\mathcal{F}_{16}$ given by Eq. (134) can be expressed in operator rather than matrix notation as follows:

$$\mathcal{F}_{16}(3, 2, 1, 0) = F_8(2, 1, 0)^{n(3)} F_4(1, 0)^{n(2)} F_2(0)^{n(1)}$$  \hspace{1cm} (137)

The above example generalizes as follows. For $\lambda = 1, 2, 3, \ldots$,

$$\mathcal{F}_{2^{\lambda+1}}(\lambda, \ldots, 2, 1, 0) = F_{2^{\lambda}}(\lambda - 1, \ldots, 2, 1, 0)^{n(\lambda) \ldots} F_4(1, 0)^{n(2)} F_2(0)^{n(1)}, \hspace{1cm} (138)$$

and

$$A_{2^{\lambda}} = \mathcal{F}_{2^{\lambda}} A_{2^{\lambda}} \mathcal{F}_{2^{\lambda}}^\dagger. \hspace{1cm} (139)$$

Thus, for $\Lambda = 1, 2, 3, \ldots$,

$$\exp(iA_{2^{\Lambda+1}}) = \mathcal{F}_{2^{\Lambda+1}} \exp(iA_{2^{\Lambda+1}}) \mathcal{F}_{2^{\Lambda+1}}^\dagger. \hspace{1cm} (140)$$

### 10.2 Separating into Strands

A consequence of diagonalizing the levels of the binary tree graph is that the tree graph is replaced by a collection of subgraphs shaped like strands. In this section, we will factor $\exp(iA_{2^{\Lambda+1}})$ into separate contributions from each of these strands.

Let us consider an example first, before dealing with the general case. Suppose $\Lambda = 5$ and consider $\exp(iA_{64})$. By studying carefully the definition Eq. (133) of $A_{64}$, one can see that

$$\exp(iA_{64}) = \prod_{s=0}^{15} \exp(iA_{64}^{(s)}), \hspace{1cm} (141)$$

where the matrices $A_{64}^{(s)}$ are defined in terms of matrices $A_{2^{\lambda}}^{(s)}$ and $B_{2^{\lambda}}^{(s)}$ as follows. The index $s$ labels the 16 “strands” that contribute to $\exp(iA_{64})$. Suppose $s \in Z_{0,15}$. Let

$$B_{2^{\lambda}}^{(s)} = \begin{bmatrix} 0_{2^{\lambda-1}} & \Delta_{2^{\lambda-1}}^{(s)} \end{bmatrix} \hspace{1cm} (142)$$

for $\lambda \in Z_{1,5}$. Let $A_{2^{\lambda}}^{(s)} = 0$ and

$$A_{2^{\lambda+1}}^{(s)} = \begin{bmatrix} A_{2^{\lambda}}^{(s)} & B_{2^{\lambda}}^{(s)\dagger} \\ B_{2^{\lambda}}^{(s)} & 0 \end{bmatrix} \hspace{1cm} (143)$$

for $\lambda \in Z_{1,5}$.

For $s = 0$, one finds
\[ A^{(0)}_{32} = (d_1|1\rangle\langle 3| + d_2|3\rangle\langle 7| + d_3|7\rangle\langle 15| + d_4|15\rangle\langle 31|) + h.c. \]

\[ B^{(0)}_{32} = d_5|31\rangle\langle 31| \]

\[ A^{(0)}_{16} = (d_1|1\rangle\langle 3| + d_2|3\rangle\langle 7| + d_3|7\rangle\langle 15|) + h.c. \]

\[ B^{(0)}_{16} = d_4|15\rangle\langle 15| \]

\[ A^{(0)}_{8} = (d_1|1\rangle\langle 3| + d_2|3\rangle\langle 7|) + h.c. \]

\[ B^{(0)}_{8} = d_3|7\rangle\langle 7| \]

\[ A^{(0)}_{4} = d_1|1\rangle\langle 3| + h.c. \]

\[ B^{(0)}_{4} = d_2|3\rangle\langle 3| \]

\[ A^{(0)}_{2} = 0 \]

\[ B^{(0)}_{2} = d_1|1\rangle\langle 1| \]

\[ (144) \]

For \( s = 1 \), one finds

\[ A^{(1)}_{32} = (d_2|2\rangle\langle 6| + d_3|6\rangle\langle 14| + d_4|14\rangle\langle 30|) + h.c. \]

\[ B^{(1)}_{32} = d_5|30\rangle\langle 30| \]

\[ A^{(1)}_{16} = (d_2|2\rangle\langle 6| + d_3|6\rangle\langle 14|) + h.c. \]

\[ B^{(1)}_{16} = d_4|14\rangle\langle 14| \]

\[ A^{(1)}_{8} = (d_2|2\rangle\langle 6|) + h.c. \]

\[ B^{(1)}_{8} = d_3|6\rangle\langle 6| \]

\[ A^{(1)}_{4} = 0 \]

\[ B^{(1)}_{4} = d_2|2\rangle\langle 2| \]

\[ A^{(1)}_{2} = 0 \]

\[ B^{(1)}_{2} = 0 \]

(145)

For \( k \in Z_{0,1} \) (and \( s = 2 + k \in Z_{2,3} \)), one finds

\[ A^{(2+k)}_{32} = (d_3|5-k\rangle\langle 13-k| + d_4|13-k\rangle\langle 29-k|) + h.c. \]

\[ B^{(2+k)}_{32} = d_5|29-k\rangle\langle 29-k| \]

\[ A^{(2+k)}_{16} = (d_3|5-k\rangle\langle 13-k|) + h.c. \]

\[ B^{(2+k)}_{16} = d_4|13-k\rangle\langle 13-k| \]

\[ A^{(2+k)}_{8} = 0 \]

\[ B^{(2+k)}_{8} = d_3|5-k\rangle\langle 5-k| \]

\[ A^{(2+k)}_{4} = A^{(2+k)}_{2} = 0 \]

\[ B^{(2+k)}_{4} = B^{(2+k)}_{2} = 0 \]

(146)

For \( k \in Z_{0,3} \) (and \( s = 4 + k \in Z_{4,7} \)), one finds

\[ A^{(4+k)}_{32} = d_4|11-k\rangle\langle 27-k| + h.c. \]

\[ B^{(4+k)}_{32} = d_5|27-k\rangle\langle 27-k| \]

\[ A^{(4+k)}_{16} = 0 \]

\[ B^{(4+k)}_{16} = d_4|11-k\rangle\langle 11-k| \]

\[ A^{(4+k)}_{8} = A^{(4+k)}_{4} = A^{(4+k)}_{2} = 0 \]

\[ B^{(4+k)}_{8} = B^{(4+k)}_{4} = B^{(4+k)}_{2} = 0 \]

(147)

For \( k \in Z_{0,7} \) (and \( s = 8 + k \in Z_{8,15} \)), one finds

\[ A^{(8+k)}_{32} = 0 \]

\[ B^{(8+k)}_{32} = d_5|23-k\rangle\langle 23-k| \]

\[ A^{(8+k)}_{16} = A^{(8+k)}_{8} = A^{(8+k)}_{4} = A^{(8+k)}_{2} = 0 \]

\[ B^{(8+k)}_{16} = B^{(8+k)}_{8} = B^{(8+k)}_{4} = B^{(8+k)}_{2} = 0 \]

(148)

The above example generalizes as follows. For \( \Lambda = 1, 2, 3, \ldots \)

\[ \exp(iA_{2^{\Lambda+1}}) = \prod_{s=0}^{2^{\Lambda+1} - 1} \exp(iA^{(s)}_{2^{\Lambda+1}}), \]

(149)
where $A_{2^{\Lambda+1}}$ is defined in terms of matrices $A_{2^{\Lambda}}^{(s)}$ and $B_{2^{\Lambda}}^{(s)}$ as follows. Suppose $s \in Z_{0,2^\Lambda-1-1}$. Let

$$B_{2^{\Lambda}}^{(s)} = \begin{bmatrix} 0_{2^{\Lambda-1}} \\ \Delta_{2^\Lambda-1}^{(s)} \end{bmatrix}$$

(150)

for $\lambda \in Z_{1,\Lambda}$. Let $A_{2^{\Lambda}}^{(s)} = 0$ and

$$A_{2^{\Lambda+1}}^{(s)} = \begin{bmatrix} A_{2^{\Lambda}}^{(s)} & B_{2^\Lambda}^{(s)^\dagger} \\ B_{2^\Lambda}^{(s)} & 0 \end{bmatrix}$$

(151)

for $\lambda \in Z_{1,\Lambda}$.

Figure 7: Illustration of strands as entries of matrix $A_{32}$. An empty cell represents a zero matrix entry. The number inside a non-empty cell is the number of the strand that owns that cell.

Figures 8, 9 and 7 are three alternative ways of visualizing strands.

It is convenient to define infinitely many strands, and to define each strand as an infinite sequence of integers. The integers in a strand label some nodes of an infinite binary tree. When we consider a finite tree, the strands are truncated to a finite length.
Figure 8: A binary tree with 32 nodes, including a dud node \( d \). Heavy blue lines mark the node strands of the tree.

For \( s = 0, 1, 2, \ldots \), let

\[
\lambda_{\text{min}}(s) = \lceil \log_2(s + 1) \rceil + 1.
\]  

Thus,

\[
\begin{array}{c|c}
  s & \lambda_{\text{min}}(s) \\
  \hline
  0 & 1 \\
  1 & 2 \\
  2, 3 & 3 \\
  4 - 7 & 4 \\
  8 - 15 & 5 \\
\end{array}
\]  

By the strand \( s \), we mean the infinite sequence

\[
2^{\lambda_{\text{min}}(s)} - s - 1, \ 2^{\lambda_{\text{min}}(s)+1} - s - 1, \ 2^{\lambda_{\text{min}}(s)+2} - s - 1, \ \ldots.
\]  

Note from our \( \Lambda = 5 \) example that, in the definition of \( A_{2^\lambda}^{(s)} \), strand \( s \) is truncated when it reaches the term:

\[
\nu_f(s, \lambda) = 2^\lambda - s - 1.
\]  

Thus, for \( \lambda \in Z_{1, \Lambda} \) and \( s \in Z_{0, 2^\Lambda - 1}, \)

\[
A_{2^\lambda}^{(s)} = \sum_{\lambda' = \lambda_{\text{min}}(s)}^{\lambda-1} d_{\lambda'} (2^{\lambda'} - s - 1)(2^{\lambda'+1} - s - 1),
\]  

and
Figure 9: Each column of this "log2 staircase" gives a different strand of the binary tree in Fig.8. Strand numbers are given at the top edge.

\[ B^{(s)}_{2^\lambda} = d_\lambda |2^\lambda - s - 1\rangle \langle 2^\lambda - s - 1| . \]  

(157)

Note that \( 2^{\Lambda-1} \) strands ( \( 2^{\Lambda-1} = \) a quarter of the number of nodes and half the number of leaves of the tree) are active (come into play) in the definition of \( A_{2^{\Lambda+1}} \). For example, 16 strands are active in the definition of \( A_{64} \), and \( A_{64} \) is associated with a tree with 64 nodes and 32 leaves.

10.3 Factoring into CSD-ready Evolutions

In this section, we will factor \( e^{iA^{(s)}_{2^\Lambda+1}} \) for each \( s \) into CSD-ready evolutions.

Let us consider an example first, before dealing with the general case. Suppose \( \Lambda = 5 \) and consider \( \exp(iA^{(s)}_{64}) \). By applying Eq. (71) repeatedly, we can factor \( e^{iA^{(s)}_{64}} \) into CSD-ready evolutions, as follows:

\[ e^{iA^{(s)}_{64}} = \prod_{\lambda=1,2,\ldots,5} \Gamma^{(s)}_{\lambda} + \mathcal{O}(g^3) , \]  

(158)

where

\[ \Gamma^{(s)}_{5} = \exp\left( i \begin{bmatrix} 0 & \mathcal{B}^{(s)}_{32} \\ \mathcal{B}^{(s)}_{32} & 0 \end{bmatrix} \right) , \]  

(159a)

\[ \Gamma^{(s)}_{4} = \exp\left( i \begin{bmatrix} 0 & \mathcal{B}^{(s)}_{16} \\ \mathcal{B}^{(s)}_{16} & 0 \end{bmatrix} \right) = \exp(i P_0 \otimes \begin{bmatrix} 0 & \mathcal{B}^{(s)}_{16} \\ \mathcal{B}^{(s)}_{16} & 0 \end{bmatrix} ) , \]  

(159b)
\[
\Gamma_3^{(s)} = \exp(i \begin{bmatrix}
0 & B_8^{(s)} \\
B_8^{(s)} & 0 \\
0 & 0
\end{bmatrix}) = \exp(i P_0^{\otimes 2} \otimes \begin{bmatrix}
0 & B_8^{(s)} \\
B_8^{(s)} & 0
\end{bmatrix}), \quad (159c)
\]

\[
\Gamma_2^{(s)} = \exp(i \begin{bmatrix}
0 & B_4^{(s)} \\
B_4^{(s)} & 0 \\
0 & 0
\end{bmatrix}) = \exp(i P_0^{\otimes 3} \otimes \begin{bmatrix}
0 & B_4^{(s)} \\
B_4^{(s)} & 0
\end{bmatrix}), \quad (159d)
\]

and

\[
\Gamma_1^{(s)} = \exp(i \begin{bmatrix}
0 & B_2^{(s)} \\
B_2^{(s)} & 0 \\
0 & 0
\end{bmatrix}) = \exp(i P_0^{\otimes 4} \otimes \begin{bmatrix}
0 & B_2^{(s)} \\
B_2^{(s)} & 0
\end{bmatrix}). \quad (159e)
\]

\(B_{2\lambda}^{(s)}\) is defined by

\[
B_{2\lambda}^{(s)} = B_{2\lambda}^{(s)} \operatorname{sinc}(\frac{A_{2\lambda}^{(s)}}{2})e^{i A_{2\lambda}^{(s)}} \quad (160)
\]

for \(\lambda \in Z_{1,5}\).

The above example generalizes as follows. For \(\Lambda = 1, 2, 3, \ldots, s \in Z_{0,2^{\Lambda-1}-1}\) and \(\lambda \in Z_{1,\Lambda}\), we have

\[
e^{i A_{2\lambda+1}^{(s)}} = \prod_{\lambda=1,2,\ldots,\Lambda}^{\Lambda} \Gamma_{\lambda}^{(s)} + O(g^3), \quad (161)
\]

\[
\Gamma_{\lambda}^{(s)} = \exp(i P_0^{\otimes \Lambda-\lambda} \otimes \begin{bmatrix}
0 & B_{2\lambda}^{(s)} \\
B_{2\lambda}^{(s)} & 0
\end{bmatrix}) \quad (162)
\]

and

\[
B_{2\lambda}^{(s)} = B_{2\lambda}^{(s)} \operatorname{sinc}(\frac{A_{2\lambda}^{(s)}}{2})e^{i A_{2\lambda}^{(s)}} \quad (163)
\]

### 10.4 Expressing the \(B_{2\lambda}^{(s)}\)’s in Bra-Ket Notation

The CSD-ready matrices \(\Gamma_{\lambda}^{(s)}\) depend on matrices \(B_{2\lambda}^{(s)}\). In this section, we will express the \(B_{2\lambda}^{(s)}\) in bra-ket notation.

Let us consider an example first, before dealing with the general case. Suppose \(\Lambda = 5\) and consider the matrices \(B_{2\lambda}^{(s)}\) that enter into the calculation of \(\exp(i A_{64}^{(s)})\). To
calculate $\mathbf{B}^{(0)}_{32}$ as defined by Eq. (160), we need to calculate $\text{sinc}(\frac{A^{(0)}_{32}}{2})e^{\frac{A^{(0)}_{32}}{2}}$. We can calculate $\text{sinc}(\frac{A^{(0)}_{32}}{2})e^{\frac{A^{(0)}_{32}}{2}}$ using its Taylor expansion at $x = 0$, provided that we can calculate all powers of the matrix $A^{(0)}_{32}$ with reasonable efficiency. This can indeed be done because $A^{(0)}_{32}$ is a sparse matrix (it’s effectively a $5 \times 5$ “tridiagonal band matrix”). Indeed, only rows and columns corresponding to nodes 1,3,7,15 and 31 are nonzero in $A^{(0)}_{32}$. It can be represented by:

$$A^{(0)}_{32} = \begin{bmatrix}
1 & & & & \\
3 & d_1 & d_2 & & \\
7 & d_2 & d_3 & & \\
15 & d_3 & d_4 & & \\
31 & & & & 
\end{bmatrix}.$$  \hspace{1cm} (164)

Hence,

$$\left(A^{(0)}_{32}\right)^2 = \begin{bmatrix}
1 & & & & \\
3 & d_1^2 & 0 & d_1 d_2 & \\
7 & d_1 d_2 & d_1^2 + d_2^2 & d_2 d_3 & \\
15 & d_2 d_3 & d_2^2 + d_3^2 & d_3 d_4 & \\
31 & & & & 
\end{bmatrix}. \hspace{1cm} (165)$$

Other powers of $A^{(0)}_{32}$ can be calculated just as easily. The pattern of entries that are zero in $(A^{(0)}_{32})^n$ (for some integer $n \geq 1$) can be easily understood by appealing to Dirac bra-ket notation. In that notation, it is clear that if a row number of $(A^{(0)}_{32})^n$ is the starting point of a staggering drunk, then the column numbers of the non-vanishing entries of that row are the possible final positions of the drunk. This, assuming that the drunk takes exactly $n$ steps, either backwards or forwards, on the 5 state runway 1, 3, 7, 15, 31. These rules imply that, at all times, the drunk will be $(n$ minus an even number) steps away from his starting point.

Note that to calculate $\mathbf{B}^{(0)}_{32}$, we don’t even need to calculate all the entries of $(A^{(0)}_{32})^n$. In $\mathbf{B}^{(0)}_{32}$, $(A^{(0)}_{32})^n$ appears pre-multiplied by $\mathbf{B}^{(0)}_{32} = d_5|31\rangle\langle 31|$. This means that we only need to calculate the last row of $(A^{(0)}_{32})^n$. For example,

$$\mathbf{B}^{(0)}_{32}(A^{(0)}_{32})^2 = d_5|31\rangle\langle 31|\{ |31\rangle d_4(15)|15\rangle d_3(7) + |31\rangle d_4(15)|15\rangle d_4(31) \}.$$ \hspace{1cm} (166)

Thus, the drunk starts at 31. After 2 steps, he must be either: back at 31, or two steps away, at 7.

The strand illustrations Figs.8, 9, and 17 can be used, in conjunction with our analogy to a drunken walker, to predict the general form of all $\mathbf{B}^{(0)}_{2k}$ that contribute
to $\exp(i A_{\lambda}^{(s)})$. One concludes that there must exist some real numbers $b_j^s$ such that the following is true. For $s = 0$,

$$
\begin{align*}
\mathcal{B}_{12}^{(0)} &= |31\rangle (b_{12}^{32}|31\rangle + i b_{15}^{32}|15\rangle + b_7^{32}|3\rangle + b_1^{32}|1\rangle) \\
\mathcal{B}_{16}^{(0)} &= |15\rangle (b_{16}^{16}|15\rangle + i b_7^{16}|7\rangle + b_3^{16}|3\rangle + i b_1^{16}|1\rangle) \\
\mathcal{B}_8^{(0)} &= |7\rangle (b_7^8|7\rangle + i b_3^8|3\rangle + b_1^8|1\rangle) \\
\mathcal{B}_{1}^{(0)} &= |3\rangle (b_3^1|3\rangle + i b_1^1|1\rangle) \\
\mathcal{B}_2^{(0)} &= |1\rangle (b_1^2|1\rangle)
\end{align*}
$$

For $s = 1$,

$$
\begin{align*}
\mathcal{B}_{32}^{(1)} &= |30\rangle (b_{32}^{32}|30\rangle + i b_{14}^{32}|14\rangle + b_6^{32}|6\rangle + i b_2^{32}|2\rangle) \\
\mathcal{B}_{16}^{(1)} &= |14\rangle (b_{16}^{16}|14\rangle + i b_6^{16}|6\rangle + b_2^{16}|2\rangle) \\
\mathcal{B}_8^{(1)} &= |6\rangle (b_6^8|6\rangle + i b_2^8|2\rangle) \\
\mathcal{B}_2^{(1)} &= |2\rangle (b_2^2|2\rangle) \\
\mathcal{B}_2^{(1)} &= 0
\end{align*}
$$

For $k \in Z_{0,1}$ (and $s = 2 + k \in Z_{2,3}$),

$$
\begin{align*}
\mathcal{B}_{32}^{(2+k)} &= |29 - k\rangle (b_{32}^{29-k}|29 - k\rangle + i b_{13-k}^{29}|13 - k\rangle + b_{5-k}^{29}|5 - k\rangle) \\
\mathcal{B}_{16}^{(2+k)} &= |13 - k\rangle (b_{16}^{13-k}|13 - k\rangle + i b_{5-k}^{16}|5 - k\rangle) \\
\mathcal{B}_8^{(2+k)} &= |5 - k\rangle (b_{5-k}^8|5 - k\rangle) \\
\mathcal{B}_2^{(2+k)} &= \mathcal{B}_2^{(2+k)} = 0
\end{align*}
$$

For $k \in Z_{0,3}$ (and $s = 4 + k \in Z_{4,7}$),

$$
\begin{align*}
\mathcal{B}_{32}^{(4+k)} &= |27 - k\rangle (b_{32}^{27-k}|27 - k\rangle + b_{11-k}^{32}|11 - k\rangle) \\
\mathcal{B}_{16}^{(4+k)} &= |11 - k\rangle (b_{16}^{11-k}|11 - k\rangle) \\
\mathcal{B}_8^{(4+k)} &= \mathcal{B}_2^{(4+k)} = \mathcal{B}_2^{(4+k)} = 0
\end{align*}
$$

For $k \in Z_{0,7}$ (and $s = 8 + k \in Z_{8,15}$),

$$
\begin{align*}
\mathcal{B}_{32}^{(8+k)} &= |23 - k\rangle (b_{32}^{23-k}|23 - k\rangle) \\
\mathcal{B}_{16}^{(8+k)} &= \mathcal{B}_8^{(8+k)} = \mathcal{B}_2^{(8+k)} = \mathcal{B}_2^{(8+k)} = 0
\end{align*}
$$

The above example generalizes as follows. For $\Lambda = 1, 2, 3, \ldots, s \in Z_{0,2^{\lambda-1}-1}$ and $\lambda \in Z_{1,\Lambda}$, we have

---

7 An important special case is when the $d_{\lambda}$ (defined by Eq. (131)) are all equal to $\sqrt{2}/2$. Appendix B gives an explicit expression for the $b_j^s$ in this case.
\[ \overline{B}^{(s)}_{2\lambda} = |2^\lambda - s - 1\rangle \sum_{\lambda' \in \mathbb{Z}_{\lambda_{\min}(s),\lambda}} i^{\theta(|\lambda - \lambda'| \text{ is odd})} b_{2^\lambda - s - 1}^{2\lambda'} = \langle 2^\lambda - s - 1 |. \] (172)

Note that \( \overline{B}^{(s)}_{2\lambda} = 0 \) if \( \lambda < \lambda_{\min}(s) \).

### 10.5 Finding SVD of \( \overline{B}^{(s)}_{2\lambda} \)'s

In this section, we find an SVD for each \( \overline{B}^{(s)}_{2\lambda} \). Luckily, such SVD's are one sided, and can be found in closed form.

As an example, assume \( \Lambda = 5 \), and consider all \( \overline{B}^{(s)}_{2\lambda} \) that enter into the calculation of \( \exp(iA_{64}) \).

**Lemma 4** The following SVD holds for some \( \rho_{31}^{32}, \theta_{31}^{32}, \theta_{15}^{32}, \theta_{7}^{32}, \theta_{3}^{32} \in \mathbb{R} \):

\[ \overline{B}^{(0)}_{32} = \rho_{31}^{32} |31\rangle \langle 31| e^{i\theta_{31}^{32} \sigma_x(4)} e^{i\theta_{15}^{32} \sigma_x(3)} e^{i\theta_{7}^{32} \sigma_x(2)} e^{i\theta_{3}^{32} \sigma_x(1)} . \] (173)

**proof:**

In this proof, the symbols \( b, \rho, \theta \) should all have a superscript of 32. We will omit this superscript to simplify the notation.

In the previous section, we showed that

\[ \overline{B}^{(0)}_{32} = |31\rangle \langle 31| + ib_{15} \langle 15 | + b_{7} \langle 7 | + ib_{3} \langle 3 | + b_{1} \langle 1 | . \] (174)

Note that

\[ 31 = (11111), \ 15 = (01111), \ 7 = (00111), \ 3 = (00011), \ 1 = (00001) . \] (175)

If we define \( \rho_3 \) and \( \theta_3 \) by

\[ \rho_3 = \sqrt{b_3^2 + b_1^2}, \ \cos \theta_3 = \frac{b_3}{\rho_3}, \ \sin \theta_3 = \frac{-b_1}{\rho_3} , \] (176)

then

\[ (ib_3, b_1) = (i\rho_3, 0) \begin{bmatrix} b_3 & -ib_1 \\ -ib_1 & b_3 \end{bmatrix} \frac{1}{\rho_3} = (i\rho_3, 0) e^{i\theta_3 \sigma_x} . \] (177)

Thus,

\[ \begin{pmatrix} b_{31} \langle 31 | + ib_{15} \langle 15 | + b_{7} \langle 7 | + \\ + ib_{3} \langle 3 | + b_{1} \langle 1 | \end{pmatrix} = \begin{pmatrix} b_{31} \langle 31 | + ib_{15} \langle 15 | + b_{7} \langle 7 | + \\ + i\rho_3 \langle 3 | \end{pmatrix} e^{i\theta_3 \sigma_x(1)} . \] (178)
If we define $\rho_7$ and $\theta_7$ by

$$
\rho_7 = \sqrt{b_7^2 + \rho_3^2}, \quad \cos \theta_7 = \frac{b_7}{\rho_7}, \quad \sin \theta_7 = \frac{\rho_3}{\rho_7},
$$

(179)

then

$$
(b_7, i\rho_3) = (\rho_7, 0) \left[ \begin{array}{cc} b_7 & i\rho_3 \\ i\rho_3 & b_7 \end{array} \right] \frac{1}{\rho_7} = (\rho_7, 0)e^{i\theta_7 \sigma_X}.
$$

(180)

Thus,

$$
\left( \begin{array}{c} b_{31} \langle 31 \mid + + i b_{15} \langle 15 \mid + \\ + b_7 \langle 7 \mid + i \rho_3 \langle 3 \mid \end{array} \right) = \left( \begin{array}{c} b_{31} \langle 31 \mid + i b_{15} \langle 15 \mid + \\ + b_7 \langle 7 \mid \end{array} \right) e^{i\bar{m}(4)\bar{m}(3)\theta_7 \sigma_X(2)}.
$$

(181)

If we define $\rho_{15}$ and $\theta_{15}$ by

$$
\rho_{15} = \sqrt{b_{15}^2 + \rho_7^2}, \quad \cos \theta_{15} = \frac{b_{15}}{\rho_{15}}, \quad \sin \theta_{15} = \frac{-\rho_7}{\rho_{15}},
$$

(182)

then

$$
(i b_{15}, \rho_7) = (i \rho_{15}, 0) \left[ \begin{array}{cc} b_{15} & -i\rho_7 \\ -i\rho_7 & b_{15} \end{array} \right] \frac{1}{\rho_{15}} = (i \rho_{15}, 0)e^{i\theta_{15} \sigma_X}.
$$

(183)

Thus,

$$
\left( \begin{array}{c} b_{31} \langle 31 \mid + + i b_{15} \langle 15 \mid + \\ + i b_{15} \langle 15 \mid + \right) = \left( \begin{array}{c} b_{31} \langle 31 \mid + i b_{15} \langle 15 \mid + \\ + b_7 \langle 7 \mid \right) e^{i\bar{m}(4)\theta_{15} \sigma_X(3)}.
$$

(184)

If we define $\rho_{31}$ and $\theta_{31}$ by

$$
\rho_{31} = \sqrt{b_{31}^2 + \rho_{15}^2}, \quad \cos \theta_{31} = \frac{b_{31}}{\rho_{31}}, \quad \sin \theta_{31} = \frac{\rho_{15}}{\rho_{31}},
$$

(185)

then

$$
(b_{31}, i\rho_{15}) = (\rho_{31}, 0) \left[ \begin{array}{cc} b_{31} & i\rho_{15} \\ i\rho_{15} & b_{31} \end{array} \right] \frac{1}{\rho_{31}} = (\rho_{31}, 0)e^{i\theta_{31} \sigma_X}.
$$

(186)

Thus,

$$
\left( \begin{array}{c} b_{31} \langle 31 \mid + i \rho_{15} \langle 15 \mid \\ + i \rho_{15} \langle 15 \mid \end{array} \right) = \rho_{31} \langle 31 \mid e^{i\theta_{31} \sigma_X(4)}.
$$

(187)

**QED**

The techniques used in the proof of the above claim can also be used to find an SVD of $B_{2}^{(s)}$ for all $s \in Z_{0,15}$ and $\lambda \in Z_{1,5}$ (These are all the $B_{2}^{(s)}$ that enter into the calculation of $\exp(iA_{64})$). One finds the following.
For \( s = 0 \),

\[
\begin{align*}
\mathcal{B}_{32}^{(0)} &= \rho_{31}^{32} |31\rangle \langle 31| e^{i\theta^a_{32} \gamma_4 (4)} e^{i\theta^a_{12} \gamma_2 (3)} e^{i\theta^a_{22} \gamma_1 (2)} e^{i\theta^a_{32} \gamma_3 (1)} \\
\mathcal{B}_{16}^{(0)} &= \rho_{15}^{16} |15\rangle \langle 15| e^{i\theta^a_{32} \gamma_4 (3)} e^{i\theta^a_{12} \gamma_2 (2)} e^{i\theta^a_{32} \gamma_3 (2)} \\
\mathcal{B}_8^{(0)} &= \rho_7^{8} |7\rangle \langle 7| e^{i\theta^a_{32} \gamma_4 (2)} e^{i\theta^a_{32} \gamma_3 (1)} \\
\mathcal{B}_4^{(0)} &= \rho_3^{4} |3\rangle \langle 3| e^{i\theta^a_{32} \gamma_1 (1)} \\
\mathcal{B}_2^{(0)} &= \rho_1^{2} |1\rangle \langle 1|
\end{align*}
\]

For \( s = 1 \),

\[
\begin{align*}
\mathcal{B}_{32}^{(1)} &= \rho_{30}^{32} |30\rangle \langle 30| e^{i\theta^a_{32} \gamma_4 (4)} e^{i\theta^a_{12} \gamma_2 (3)} e^{i\theta^a_{32} \gamma_3 (2)} \\
\mathcal{B}_{16}^{(1)} &= \rho_{14}^{16} |14\rangle \langle 14| e^{i\theta^a_{32} \gamma_4 (3)} e^{i\theta^a_{12} \gamma_2 (2)} \\
\mathcal{B}_8^{(1)} &= \rho_6^{8} |6\rangle \langle 6| e^{i\theta^a_{32} \gamma_2 (2)} \\
\mathcal{B}_4^{(1)} &= \rho_2^{4} |2\rangle \langle 2| \\
\mathcal{B}_2^{(1)} &= 0
\end{align*}
\]

For \( k \in \mathbb{Z}_{0,1} \) (and \( s = 2 + k \in \mathbb{Z}_{2,3} \)),

\[
\begin{align*}
\mathcal{B}_{32}^{(2+k)} &= \rho_{29-k}^{32} |29-k\rangle \langle 29-k| e^{i\theta^a_{32} \gamma_4 (4)} e^{i\theta^a_{12} \gamma_2 (3)} e^{i\theta^a_{32} \gamma_3 (2)} \\
\mathcal{B}_{16}^{(2+k)} &= \rho_{13-k}^{16} |13-k\rangle \langle 13-k| e^{i\theta^a_{32} \gamma_2 (3)} e^{i\theta^a_{32} \gamma_3 (2)} \\
\mathcal{B}_8^{(2+k)} &= \rho_{5-k}^{8} |5-k\rangle \langle 5-k|
\end{align*}
\]

For \( k \in \mathbb{Z}_{0,3} \) (and \( s = 4 + k \in \mathbb{Z}_{4,7} \)),

\[
\begin{align*}
\mathcal{B}_{32}^{(4+k)} &= \rho_{27-k}^{32} |27-k\rangle \langle 27-k| e^{i\theta^a_{32} \gamma_4 (4)} \\
\mathcal{B}_{16}^{(4+k)} &= \rho_{11-k}^{16} |11-k\rangle \langle 11-k| \\
\mathcal{B}_8^{(4+k)} &= \mathcal{B}_4^{(4+k)} = \mathcal{B}_2^{(4+k)} = 0
\end{align*}
\]

For \( k \in \mathbb{Z}_{0,7} \) (and \( s = 8 + k \in \mathbb{Z}_{8,15} \)),

\[
\begin{align*}
\mathcal{B}_{32}^{(8+k)} &= \rho_{23-k}^{32} |23-k\rangle \langle 23-k| \\
\mathcal{B}_{16}^{(8+k)} &= \mathcal{B}_8^{(8+k)} = \mathcal{B}_4^{(8+k)} = \mathcal{B}_2^{(8+k)} = 0
\end{align*}
\]

10.6 **Expressing the \( \Gamma_{\lambda}^{(s)} \)'s as EG Circuits**

In this section, we express each \( \Gamma_{\lambda}^{(s)} \) as an EG (elementary gates) circuit.

Let us consider an example first, before dealing with the general case. Suppose \( \Lambda = 5 \).

Consider \( s = 0 \).
Plugging the SVD of $\mathcal{B}^{(0)}_{32}$, given by Eq. (188), into the definition of $\Gamma_{5}^{(0)}$, given by Eq. (159a), one finds

$$
\Gamma_{5}^{(0)} = 
\begin{bmatrix}
U_{32}^{\dagger} & I_{32}
\end{bmatrix}
\cdot 
\begin{bmatrix}
\rho^{(196b)}_{31} & \sigma_{X}\langle 5 \rangle \langle n(4) n(3) n(2) n(1) \rangle
\end{bmatrix}
\begin{bmatrix}
U_{32} & I_{32}
\end{bmatrix}
$$

where $U_{32}$ is defined by

$$
U_{32}(4, 3, 2, 1, 0) = e^{i \theta_{31}^{(196b)} \sigma_{X}(4)} e^{\bar{\psi}_{4}^{(32)} \sigma_{X}(3)} e^{\bar{\psi}_{4}^{(32)} \sigma_{X}(2)} e^{\bar{\psi}_{4}^{(32)} \sigma_{X}(1)} .
$$

Eq. (193b) simplifies to

$$
\Gamma_{5}^{(0)} = e^{-i \bar{\psi}_{4}^{(32)} \sigma_{X}(4)} e^{i \bar{\psi}_{4}^{(32)} \sigma_{X}(3)} e^{i \bar{\psi}_{4}^{(32)} \sigma_{X}(2)} e^{i \bar{\psi}_{4}^{(32)} \sigma_{X}(1)} \cdot
$$

Plugging the SVD of $\mathcal{B}^{(0)}_{16}$, given by Eq. (188), into the definition of $\Gamma_{4}^{(0)}$, given by Eq. (159b), one finds

$$
\Gamma_{4}^{(0)} = 
\begin{bmatrix}
U_{16}^{\dagger} & I_{16}
\end{bmatrix}
\cdot 
\begin{bmatrix}
\rho^{(196b)}_{15} & \sigma_{X}\langle 5 \rangle \langle n(4) n(3) n(2) n(1) \rangle
\end{bmatrix}
\begin{bmatrix}
U_{16} & I_{16}
\end{bmatrix}
$$

where $U_{16}$ is defined by

$$
U_{16}(3, 2, 1, 0) = e^{i \theta_{15}^{(196b)} \sigma_{X}(3)} e^{\bar{\psi}_{4}^{(32)} \sigma_{X}(2)} e^{\bar{\psi}_{4}^{(32)} \sigma_{X}(1)} .
$$

Eq. (196b) simplifies to

$$
\Gamma_{4}^{(0)} = e^{-i \bar{\psi}_{4}^{(32)} \sigma_{X}(4)} e^{i \bar{\psi}_{4}^{(32)} \sigma_{X}(3)} \cdot
$$

Similarly, one finds

$$
\Gamma_{3}^{(0)} = 
\begin{bmatrix}
U_{8}^{\dagger} & I_{8}
\end{bmatrix}
\cdot 
\begin{bmatrix}
\rho^{(196b)}_{6} & \sigma_{X}\langle 7 \rangle \langle n(4) n(3) n(2) n(1) \rangle
\end{bmatrix}
\begin{bmatrix}
U_{8} & I_{8}
\end{bmatrix}
$$

where

$$
\Gamma_{3}^{(0)}(2, 1, 0) = e^{i \bar{\psi}_{4}^{(32)} \sigma_{X}(3)} e^{\bar{\psi}_{4}^{(32)} \sigma_{X}(2)} e^{\bar{\psi}_{4}^{(32)} \sigma_{X}(1)} .
$$
\[ U_b(2, 1, 0) = e^{i\theta_5^b \sigma_X(2)} e^{i\theta_3^b \sigma_X(1)}. \]  

Eq. [199b] simplifies to

\[ \Gamma_3^{(0)} = e^{-i\pi (3) n(2) \theta_3^b \sigma_X(1)} e^{-i\pi (3) \theta_3^b \sigma_X(2)} \odot \left\{ e^{i\pi (5) \rho_3^b \sigma_X(3)n(2)n(1)n(0)} \right\}. \]  

Similarly, one finds

\[ \Gamma_2^{(0)} = \left[ U_4^\dagger I_4 \right] \oplus \left[ U_4 I_4 \right] \oplus \left[ U_4 (1, 0) \Gamma (2) e^{i\pi (5) \rho_3^b \sigma_X(2)n(1)n(0)} U_4 (1, 0) \Gamma (2) \right], \]  

where

\[ U_4(1, 0) = e^{i\theta_3^b \sigma_X(1)}. \]  

Eq. [202b] simplifies to

\[ \Gamma_2^{(0)} = e^{-i\pi (2) \theta_3^b \sigma_X(1)} \odot \left\{ e^{i\pi (5) \rho_3^b \sigma_X(2)n(1)n(0)} \right\}. \]  

Similarly, one finds

\[ \Gamma_1^{(0)} = e^{i\rho_3^b \sigma_X(1)} \odot \left\{ e^{i\rho_3^b \sigma_X(2)n(1)n(0)} \right\}. \]  

To summarize, we have shown that:

\[ \Gamma_5^{(0)} = e^{-i\pi (5) \rho_3^b \sigma_X(1)} e^{-i\pi (5) \rho_3^b \sigma_X(2)} e^{-i\pi (5) \rho_3^b \sigma_X(3)} e^{-i\pi (5) \rho_3^b \sigma_X(4)} \odot \left\{ e^{i\rho_3^b \sigma_X(5)n(4)n(3)n(2)n(1)n(0)} \right\}. \]

\[ \Gamma_4^{(0)} = e^{-i\pi (3) \rho_3^b \sigma_X(2)} e^{-i\pi (3) \rho_3^b \sigma_X(3)} e^{-i\pi (3) \rho_3^b \sigma_X(4)} \odot \left\{ e^{i\rho_3^b \sigma_X(3)n(2)n(1)n(0)} \right\}. \]

\[ \Gamma_3^{(0)} = e^{-i\pi (2) \rho_3^b \sigma_X(1)} e^{-i\pi (2) \rho_3^b \sigma_X(2)} e^{-i\pi (2) \rho_3^b \sigma_X(3)} \odot \left\{ e^{i\rho_3^b \sigma_X(2)n(1)n(0)} \right\}. \]

Fig 10 represents the SEOs of Eq. [206] as circuits.

Using the same method that we used for \( s = 0 \), we find for \( s = 1 \):

\[ \Gamma_5^{(1)} = e^{-i\pi (5) \rho_3^b \sigma_X(2)} e^{-i\pi (5) \rho_3^b \sigma_X(3)} e^{-i\pi (5) \rho_3^b \sigma_X(4)} \odot \left\{ e^{i\rho_3^b \sigma_X(5)n(4)n(3)n(2)n(1)n(0)} \right\}. \]

\[ \Gamma_4^{(1)} = e^{-i\pi (3) \rho_3^b \sigma_X(2)} e^{-i\pi (3) \rho_3^b \sigma_X(3)} \odot \left\{ e^{i\rho_3^b \sigma_X(3)n(2)n(1)n(0)} \right\}. \]

\[ \Gamma_3^{(1)} = e^{-i\pi (2) \rho_3^b \sigma_X(1)} \odot \left\{ e^{i\rho_3^b \sigma_X(2)n(1)n(0)} \right\}. \]

\[ \Gamma_2^{(1)} = e^{i\rho_3^b \sigma_X(2)n(1)n(0)} \]

\[ \Gamma_1^{(1)} = 1. \]  

(207)
Fig. 10: $\Gamma_{s\lambda}$'s for $s = 0$.

Fig. 11 represents the SEOs of Eq. (207) as circuits.

For $s = 2$, we find
\begin{align*}
\Gamma_5^{(2)} &= e^{-i\pi(5\bar{m}(4)\bar{\theta}_{13}^2\sigma_{X}(3))} e^{-i\pi(5\bar{m}(4)\bar{\sigma}_{32}\sigma_{X}(4))} \odot \{ e^{i\rho_{32}\sigma_{X}(5)n(4)n(3)n(2)\bar{m}(1)n(0)} \} \\
\Gamma_4^{(2)} &= e^{-i\pi(4)\bar{m}(4)\bar{\theta}_{13}^2\sigma_{X}(3)} \odot \{ e^{i\rho_{32}\sigma_{X}(4)n(3)n(2)\bar{m}(1)n(0)} \} \\
\Gamma_3^{(2)} &= e^{i\pi(5)\bar{m}(4)\rho_{32}\sigma_{X}(3)n(2)\bar{m}(1)n(0)} \\
\Gamma_2^{(2)} &= \Gamma_1^{(2)} = 1.
\end{align*}

The $\Gamma_{s\lambda}$ for $s = 3$ are identical to those for $s = 2$, except that one must replace $n(0)$ by $\bar{n}(0)$ in them. The angle parameters (i.e., $\rho_k, \theta_k, b_k$) also change. Fig. 12 represents the SEOs of Eq. (208) as circuits.

For $s = 4$, we find
\begin{align*}
\Gamma_5^{(4)} &= e^{-i\pi(5)\bar{m}(4)\bar{\theta}_{13}^2\sigma_{X}(4)} \odot \{ e^{i\rho_{32}\sigma_{X}(5)n(4)n(3)\bar{m}(2)\bar{m}(1)n(0)} \} \\
\Gamma_4^{(4)} &= e^{i\pi(4)\bar{m}(4)\rho_{32}\sigma_{X}(4)n(3)\bar{m}(2)n(1)n(0)} \\
\Gamma_3^{(4)} &= \Gamma_2^{(4)} = \Gamma_1^{(4)} = 1.
\end{align*}

The $\Gamma_{s\lambda}$ for $s = 5, 6, 7$ are identical to those for $s = 4$, except that we must replace $n(1)n(0)$ by $n(1)\bar{m}(0)$, $\bar{n}(1)n(0)$, and $\bar{n}(1)\bar{m}(0)$, respectively, in them. The angle parameters also change. Fig. 13 represents the SEOs of Eq. (209) as circuits.

For $s = 8$, we find
\begin{align*}
\Gamma_5^{(8)} &= e^{i\rho_{32}\sigma_{X}(5)n(4)\bar{m}(3)n(2)n(1)n(0)} \\
\Gamma_4^{(8)} &= \Gamma_3^{(8)} = \Gamma_2^{(8)} = \Gamma_1^{(8)} = 1.
\end{align*}

The $\Gamma_{s\lambda}$ for $s = 9, 10, \ldots, 15$ are identical to those for $s = 8$, except that we must
Figure 11: $\Gamma_{\lambda}^{(s)}$s for $s = 1$. $\Gamma_{(1)}^{(1)}$ is not shown because it equals 1.

replace $n(2)n(1)n(0)$ by the appropriate projection operator in them. The angle parameters also change. Fig[14] represents the SEOs of Eq.(210) as circuits.

The above example generalizes as follows. Suppose $\Lambda = 1, 2, 3, \ldots$ and $s \in Z_{2^{\Lambda-1}-1}$. Then $\Gamma_{\lambda}^{(s)} = 0$ if $\lambda < \lambda_{\text{min}}(s)$ since for such $\lambda$, $\mathcal{F}_{2\lambda}^{(s)} = 0$. For $\lambda \in Z_{\lambda_{\text{min}}(s), \Lambda}$, there exist $\phi_{\lambda,j}^{(s)}, \theta_{\lambda}^{(s)} \in \mathbb{R}$ such that

$$\Gamma_{\lambda}^{(s)} = \prod_{j=\lambda-\lambda_{\text{min}}(s)-1, \ldots, 2, 1, 0} \{ e^{\overline{\pi}(\lambda)\overline{\pi}(\lambda-1)\ldots\overline{\pi}(\lambda-j)\phi_{\lambda,j}^{(s)}\sigma_x(\lambda-j-1)} \} \odot \{ e^{\overline{\pi}(\lambda)\overline{\pi}(\lambda-1)\ldots\overline{\pi}(\lambda+1)\theta_{\lambda}^{(s)}\sigma_x(\lambda)\mathcal{F}_{\text{bin}(s)}(\lambda-1, \ldots, 1, 0)} \}.$$ (211)

In Eq.(211), we use $\text{bin}(s) = (s_{\lambda-1}s_{\lambda-2} \ldots s_1s_0)$ to denote the binary representation of $s$, and $\overline{\text{bin}}(s) = (\overline{s}_{\lambda-1}\overline{s}_{\lambda-2} \ldots \overline{s}_1\overline{s}_0)$ to denote its bitwise negation.

10.7 Weaving the Strands

In this section, we finally achieve our goal of compiling $e^{iH_{tr}}$.

Suppose $\vec{\alpha}$ and $\vec{\beta}$ are n-tuples (ordered sets), not necessarily of the same length, of distinct qubit positions. Assume that these two n-tuples are disjoint (if treated as non-ordered sets). Let $\pi$ and $\pi'$ be commuting projection operators acting on qubits $\vec{\beta}$. Let $M$ be any operator acting on qubits $\vec{\alpha}$. A simple identity that will be useful in what follows is

$$M(\vec{\alpha})^\pi(\vec{\beta}) M(\vec{\alpha})^\pi'(\vec{\beta}) = M(\vec{\alpha})^{\pi(\vec{\beta})+\pi'(\vec{\beta})}.$$ (212)

This identity allows us to combine operations that are being needlessly performed separately.
Define $\Gamma^{(s)}$ for $s \in Z_{0,2\Lambda-1-1}$ by

$$\Gamma^{(s)} = e^{i A^{(s)}_{2\Lambda+1}} = \prod_{\lambda=1,2,\ldots,\Lambda} \Gamma^{(s)}(\lambda) + O(g^3), \quad (213)$$

where we have used Eq. (161).

Note that the $\Gamma^{(s)}$ for different strands commute, because any two different strands “live” on disjoint sets of nodes. The main point of this section is that many operations can be combined if we multiply the $\Gamma^{(s)}$’s judiciously.

For definiteness, consider our usual example of $\Lambda = 5$.

Note from Fig. 8, that strands can be grouped into equivalence classes \{0\}, \{1\}, $Z_{2,3}$, $Z_{4,7}$ and $Z_{8,15}$. All members of the same equivalence class start and end at the same tree levels. They also have the same strengths $d_i$’s for transitions between levels. One could say that they are identical in every respect except that the names of their nodes differ.

Now consider Fig. 14, which gives circuits for the $\Gamma^{(s)}$’s, for strand class $Z_{8,15}$. This figure does not tell us what are the specific angles for the qubit rotations of the type $R_x$. But from our observation that equivalent strands are “identical except for node relabelling”, it is clear that any qubit rotation of type $R_x$ in a strand $s_1$ is identical to the “analogous” one in any strand $s_2$ of the same class. Therefore, we can apply identity Eq. (212) to combine the $\Gamma^{(s)}$ of all the strands of class $Z_{8,15}$:
Figure 13: $\Gamma^{(s)}_\lambda$’s for $s = 4$ to 7. $\Gamma^{(s)}_1$, $\Gamma^{(s)}_2$ and $\Gamma^{(s)}_3$ are not shown because they equal 1.

\[
\prod_{s \in \mathbb{Z}_{8,15}} \Gamma^{(s)} = \begin{array}{c}
\end{array}.
\] (214a)

Likewise, by applying identity Eq.(212) to Fig.(13), one finds that

\[
\prod_{s \in \mathbb{Z}_{4,7}} \Gamma^{(s)} = \begin{array}{c}
\end{array}.
\] (214b)

Finally, by applying identity Eq.(212) to Fig.(12), one finds
Figure 14: $\Gamma_{\lambda}^{(s)}$s for $s = 8$ to 15. $\Gamma_{1}^{(s)}$, $\Gamma_{2}^{(s)}$, $\Gamma_{3}^{(s)}$ and $\Gamma_{4}^{(s)}$ are not shown because they equal 1.

\[ \prod_{s \in \mathbb{Z}_{2,3}} \Gamma^{(s)} = \ldots \] (214c)

10.8 Complexity of $e^{iH_{tr}}$ Compilation

In this section, we will show that our SEO for $e^{iH_{tr}}$ contains $O(\Lambda^4)$ CNOTs, in the case of balanced binary NAND trees.

If $S$ represents a SEO or its associated circuit, let $C(S)$ denote its circuit complexity. We will evaluate the circuit complexity of our SEO for $e^{iH_{tr}}$ by counting the number of control vertices in its quantum circuit.

Assume a tree with $2^{\Lambda+1}$ nodes. According to Eq.(140),

\[ C(e^{iH_{tr}}) = 2C(F_{2^{\Lambda+1}}) + C(exp(iA_{2^{\Lambda+1}})) \] . (215)

According to Eq.(138),

\[ C(F_{2^{\Lambda+1}}) = \sum_{\lambda=1}^{\Lambda} C(F_{2\lambda}(\lambda - 1, \ldots, 2, 1, 0)_{n(\lambda)}) \] . (216)
An exchange operator $E_4(\alpha, \beta)$ can be expressed as product of 3 CNOTs. So a singly controlled exchange operator has six control vertices. This observation and Eq. (130) imply that

$$\mathcal{C}(F_{2\lambda}^{(\lambda)}) = 6\lambda - 5. \quad (217)$$

Combining Eqs. (216) and (217) yields

$$\mathcal{C}(F_{2\Lambda+1}) = \mathcal{O}(\Lambda^2). \quad (218)$$

According to Eqs. (149) and (213),

$$\mathcal{C}(\exp(i\mathcal{A}_{2\Lambda+1})) = \mathcal{C}(\prod_{s=0}^{2^{\Lambda+1}-1} \Gamma(\lambda)) = \mathcal{C}(\Gamma(0)) + \sum_{\sigma \in \mathbb{Z}_{0,\Lambda-2}} \mathcal{C}(\prod_{s=2^\sigma,2^\sigma+1,...,2^\sigma+1-1} \Gamma(\lambda)). \quad (219)$$

Here $\sigma$ labels the different strand equivalence classes. According to Eqs. (214), for $\sigma \in \mathbb{Z}_{0,\Lambda-2},$

$$\mathcal{C}(\prod_{s=2^\sigma,2^\sigma+1,...,2^\sigma+1-1} \Gamma(\lambda)) = \sum_{\lambda=1}^{\Lambda} \left\{ \mathcal{C}(\Gamma^{(2^\sigma)}(\lambda)) - \sigma \right\}. \quad (220)$$

According to Eq. (211) and Figs. 10 to 14

$$\mathcal{C}(\Gamma^{(s)}(\lambda)) = \Lambda - 2^{\Lambda-\lambda_{\min}(s)} - 1 \sum_{j=0}^{\lambda - \lambda_{\min}(s)-1} \{j+1\}. \quad (221)$$

Combining Eqs. (219), (220), (221), and doing some algebra, yields

$$\mathcal{C}(\exp(i\mathcal{A}_{2\Lambda+1})) = \mathcal{O}(\Lambda^4). \quad (222)$$

Now we can use Eq. (215), (218), and (222), to conclude that

$$\mathcal{C}(e^{iH_{\text{tr}}}) = \mathcal{O}(\Lambda^4). \quad (223)$$

11 Compiling Input Graph (“Oracle”)

In this section, we will show how to compile the evolution operator $e^{iH_{\text{in}}}$ for an input graph (“oracle”).

52
Suppose $\Lambda = 1, 2, 3 \ldots $. Consider a tree with $N_{\text{lvs}} = 2^\Lambda$ leaves, with leaf inputs $x_k \in \text{Bool}$ for $k \in Z_{0,2^\Lambda-1}$. The input Hamiltonian for such a tree is:

$$H_{\text{in}} = g \sum_{\mathbf{x} \in \text{Bool}^\Lambda} x_{\mathbf{x}} P_{\mathbf{x}},$$

(224b)

where $g \in \mathbb{R}$, and $x_{\mathbf{x}}$ is defined to equal $x_{\text{dec}(\mathbf{x})}$.

Define operators $U$ and $\pi$ by

$$U = e^{ig\sigma X}, \quad \pi = \sum_{\mathbf{b} \in \text{Bool}^\Lambda} x_{\mathbf{b}} P_{\mathbf{b}}.$$

(225)

Let

$$\mathbf{\xi} = (\Lambda - 1, \ldots, 1, 0).$$

(226)

Then

$$e^{iH_{\text{in}}} = [U(\Lambda)]^{\pi(\mathbf{\xi})}$$

(227a)

$$= \sigma X(\alpha)^{\pi(\mathbf{\xi})} U(\Lambda)^{n(\alpha)} \sigma X(\alpha)^{\pi(\mathbf{\xi})} |0\rangle_\alpha.$$

(227b)

Here $\alpha$ is an ancilla qubit. To get Eq. (227b), we did the same as we did in Eq. (40): we replaced the multiply controlled $U(2)$ operator by a singly controlled one, and two MCNOTs. So, to compile $e^{iH_{\text{in}}}$, all we need to do is to compile $\sigma X(\alpha)^{\pi(\mathbf{\xi})}$.

The operator $\sigma X(\alpha)^{\pi(\mathbf{\xi})}$ appears frequently in the quantum computation literature, but usually in a different guise. It’s more common to denote the function $x$ in Eq. (225) by a more standard name for a function, such as $f_\Lambda : \text{Bool}^\Lambda \rightarrow \text{Bool}$. In this notation, the operator $\pi$ of Eq. (225) becomes

$$\pi = \sum_{\mathbf{x}' \in \text{Bool}^\Lambda} f_\Lambda(\mathbf{x}') P_{\mathbf{x'}}.$$

(228)

Thus,
\[ \sigma_X(\alpha) \pi(\vec{\xi}) |y\rangle_a |\vec{x}\rangle_{\vec{\xi}} = |y \oplus f_A(\vec{x})\rangle_a |\vec{x}\rangle_{\vec{\xi}}. \]  

(229)

The best general way of compiling \( \sigma_X(\alpha) \pi(\vec{\xi}) \) is not known. It is known that there are some \( f_A() \) for which \( \sigma_X(\alpha) \pi(\vec{\xi}) \) has complexity \( O(2^\Lambda) = O(N_{\text{lv}s}) \). In Appendix C we give a compilation for a special case, that of a “banded oracle” (an oracle for which the ordered set of NAND formula inputs \((x_k)_{\forall k}\) has a fixed number of bands of ones). Our compilation of the banded oracle evolution operator has complexity \( O(\Lambda^2) \).

12 Overall Circuit Complexity

The following table summarizes the circuit complexity of the various compilations that have been presented in this paper:

| evolution operator | description | number of CNOTs in SEO | exact? |
|-------------------|-------------|-------------------------|-------|
| \( e^{iH_{tr}} \) | balanced binary NAND tree with \( 2^\Lambda = N_{\text{lv}s} \) leaves | \( O(\Lambda^4) \) | no |
| \( e^{iH_{lp}} \) | loop of length \( N_S \) | \( O(N_S) \) | yes |
| \( e^{iH_{gl}} \) | glue connecting tree and loop | \( O(\Lambda) \) | yes |
| \( e^{iH_{cut}} \) | cuts a loop open | \( O(\Lambda) \) | yes |
| \( e^{iH_{line}} \) | line of length \( N_S \) | \( O(N_{Bl}^2) \) | no |
| \( e^{iH_{in}} \) | oracle, with \( 2^\Lambda = N_{\text{lv}s} \) inputs | Could reach \( O(N_{\text{lv}s}) \). \( O(\Lambda^2) \) for banded oracle. | |

Henceforth, we will say that a SEO is tractable if it has \( O(\Lambda^k) \) CNOTs, for some \( k \geq 0 \).

Our plan is to combine \( e^{iH_{\text{bulk}}} \) and \( e^{iH_{\text{corr}}} \) via Trotterized Suzuki. Let \( N_{\exp} \) be the number of times \( e^{iH_{\text{bulk}}} \) or \( e^{iH_{\text{corr}}} \) appear in the Trotterized Suzuki product that approximates \( e^{i(H_{\text{bulk}}+H_{\text{corr}})} \). According to Eq. (39), if we want to evaluate a NAND formula in time \( \approx O((N_{\text{lv}s})^{\frac{1}{2}+\epsilon}) \), then

\[ N_{\exp} \approx O((N_{\text{lv}s})^{\frac{1}{2}+\epsilon}). \]  

(230)

Thus

\[ C(e^{i(H_{\text{bulk}}+H_{\text{corr}})}) = N_{\exp} C(e^{iH_{\text{bulk}}} e^{iH_{\text{corr}}}) \] 

\[ \approx O((N_{\text{lv}s})^{\frac{1}{2}+\epsilon}) C(e^{iH_{\text{bulk}}} e^{iH_{\text{corr}}}) . \]  

(231a)

(231b)

If \( e^{iH_{\text{bulk}}} e^{iH_{\text{corr}}} \) is tractable, then \( C(e^{i(H_{\text{bulk}}+H_{\text{corr}})}) = O((N_{\text{lv}s})^{\frac{1}{2}+\epsilon}) \).
A Appendix: Exponential of Anti-block Diagonal Matrix

Lemma 5 Suppose $F \in \mathbb{C}^{n \times n}$. Then

$$\exp(i \begin{bmatrix} 0 & F^\dagger \\ F & 0 \end{bmatrix}) = \begin{bmatrix} \cos(\sqrt{F^\dagger F}) & i \frac{\sin(\sqrt{F^\dagger F})}{\sqrt{F^\dagger F}} F^\dagger \\ i \frac{\sin(\sqrt{FF^\dagger})}{\sqrt{FF^\dagger}} F & \cos(\sqrt{FF^\dagger}) \end{bmatrix}.$$ (232)

proof: Note that

$$\begin{bmatrix} 0 & F^\dagger \\ F & 0 \end{bmatrix} = \sigma_- \otimes F + \sigma_+ \otimes F^\dagger.$$ (233)

Since $\sigma_+ = |0\rangle \langle 1|$ and $\sigma_- = |1\rangle \langle 0|$, $\sigma_+^2 = \sigma_-^2 = 0$. Thus, when we raise the right hand side of Eq.(233) to a power, only terms proportional to an alternating sequence of $\sigma_+$ and $\sigma_-$ survive. For $n = 1, 2, 3, \ldots$, one gets

$$(\sigma_- \otimes F + \sigma_+ \otimes F^\dagger)^{2n} = (\sigma_+ \sigma_-)^n \otimes (F^\dagger F)^n + (\sigma_- \sigma_+)^n \otimes (FF^\dagger)^n$$ (234a)

$$= P_0 \otimes (F^\dagger F)^n + P_1 \otimes (FF^\dagger)^n,$$ (234b)

and

$$(\sigma_- \otimes F + \sigma_+ \otimes F^\dagger)^{2n+1} = P_0 \sigma_+ \otimes (F^\dagger F)^n F^\dagger + P_1 \sigma_- \otimes (FF^\dagger)^n F$$ (235a)

$$= \sigma_+ \otimes (F^\dagger F)^n F^\dagger + \sigma_- \otimes (FF^\dagger)^n F.$$ (235b)

Thus

$$\sum_{n=0}^{\infty} \frac{(i)^{2n}}{(2n)!} (\sigma_- \otimes F + \sigma_+ \otimes F^\dagger)^{2n} = P_0 \otimes \cos(\sqrt{F^\dagger F}) + P_1 \otimes \cos(\sqrt{FF^\dagger}),$$ (236)

and

$$\sum_{n=0}^{\infty} \frac{(i)^{2n+1}}{(2n + 1)!} (\sigma_- \otimes F + \sigma_+ \otimes F^\dagger)^{2n+1} = \sigma_+ \otimes i \frac{\sin(\sqrt{F^\dagger F})}{\sqrt{F^\dagger F}} F^\dagger + \sigma_- \otimes i \frac{\sin(\sqrt{FF^\dagger})}{\sqrt{FF^\dagger}} F.$$ (237)

QED
Appendix: Special Case $d_{\lambda} = \sqrt{2}g$ for all $\lambda$

An important special case is when $d_{\lambda} = \sqrt{2}g$ for all $\lambda$, where $d_{\lambda}$ is defined by Eq. (131). This appendix will discuss some idiosyncrasies of this special case.

When the $d_{\lambda}$ are all equal, it is possible to make a slight redefinition of $\Gamma^\lambda$ in Eq.(111) so that the decomposition of $e^{iA_{2\Lambda+1}}$ into $\Gamma_{2\lambda}$ given by Eq.(110) is good to order $O(g^4)$ instead of $O(g^3)$. We show this next for $\Lambda = 3$. Generalization to higher $\Lambda$ will be obvious.

Lemma 6 Let

$$f = 1 + \frac{g^2}{6}.$$  \hspace{1cm} (238)

Then

$$e^{iA_{16}} = \Gamma_1^{<3>} \Gamma_2^{<3>} \Gamma_3^{<3>} + O(g^4),$$  \hspace{1cm} (239)

where

$$\Gamma_3^{<3>} = \exp(i \left[ \begin{array}{cc} 0 & \mathcal{B}_8 \\ \mathcal{B}_8 & 0 \end{array} \right] ) ,$$  \hspace{1cm} (240a)

$$\Gamma_2^{<3>} = \exp(i f \left[ \begin{array}{cc} 0 & \mathcal{B}_4 \\ \mathcal{B}_4 & 0 \end{array} \right] ) ,$$  \hspace{1cm} (240b)

$$\Gamma_1^{<3>} = \exp(i f \left[ \begin{array}{cc} 0 & \mathcal{B}_2 \\ \mathcal{B}_2 & 0 \end{array} \right] ) ,$$  \hspace{1cm} (240c)

and

$$\mathcal{B}_{2\lambda} = \mathcal{B}_{2\lambda} \sin\left(\frac{A_{2\lambda}}{2}\right)e^{\frac{A_{2\lambda}}{2}} , \text{ for } \lambda \in Z_{1,3} .$$  \hspace{1cm} (241)

proof: According to Eq.(67),

$$e^{iA_{16}} = \exp(i \left[ \begin{array}{cc} A_8 & \mathcal{B}_8 \\ \mathcal{B}_8 & 0 \end{array} \right] ) \hspace{1cm} (242)$$

$$= \exp(i \left[ \begin{array}{cc} A_8 & 0 \\ 0 & 0 \end{array} \right] ) \exp\left[ \begin{array}{cc} T_1 + T_2 & \frac{i}{6} \mathcal{B}_8 \mathcal{B}_8 \\ \frac{i}{6} \mathcal{B}_8 \mathcal{B}_8 & -i \mathcal{B}_8 \mathcal{B}_8 \end{array} \right] \right) + O(g^5) .$$  \hspace{1cm} (243)

One finds that
\[ T_1 = \frac{i g^2}{6} \begin{bmatrix} 0 & B_4 \\ B_4 & 0 \end{bmatrix}, \quad T_2 = O(g^4), \quad (244) \]

and

\[ B_8 A_8 B_8 = 0. \quad (245) \]

Therefore,

\[ e^{i A_{16}} = \exp(i \begin{bmatrix} A_8 & 0 \\ 0 & 0 \end{bmatrix}) \exp(i \begin{bmatrix} \frac{g^2}{6} & B_4 \\ B_4 & 0 \end{bmatrix} \overline{B_8} \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}) + O(g^4) \quad (246) \]

\[ = \exp(i \begin{bmatrix} A_4 & fB_4 \\ fB_4 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}) \Gamma_{<3>}^3 \quad (247) \]

To go from Eq. (246) to Eq. (247), we used \( A_8 = 0 \). Furthermore, \( A_4 = \begin{bmatrix} A_4 & B_1 \\ B_4 & 0 \end{bmatrix} \). Indeed, \( g^2 f \approx g^2 \).

QED

Another useful consequence of \( d_{\lambda} = \sqrt{2} g \) for all \( \lambda \) is that in this case it is easy to calculate the \( b_k^j \) coefficients used in Section 10.4. Indeed, observe that in this case all non-zero entries of \( A_{2\lambda} \) equal \( \sqrt{2} g \). Define

\[ b = g \sqrt{2} \text{sinc}(\frac{g \sqrt{2}}{2}) e^{i \frac{g \sqrt{2}}{2}}. \quad (250) \]

Let \( b_r \) and \( b_i \) denote the real and imaginary parts of \( b \), respectively. Now we can use the following rule. In Section 10.4 in the bra-ket expansions of \( \Phi^{(4)}_{2\lambda} \), replace, all \( b_k^j \) that are preceded by an imaginary \( i \), by \( b_i \), and replace, all \( b_k^j \) that are NOT preceded by an imaginary \( i \), by \( b_r \). For example, in the example of Lemma 4, one has
\[
\rho_3 = \sqrt{b_i^2 + b_r^2} = |b|, \quad \cos \theta_3 = \frac{b_i}{|b|}, \quad \sin \theta_3 = \frac{b_r}{|b|}, \quad (251)
\]

\[
\rho_7 = \sqrt{b_i^2 + |b|^2}, \quad \cos \theta_7 = \frac{b_i}{\rho_7}, \quad \sin \theta_7 = \frac{\rho_3}{\rho_7}, \quad (252)
\]

\[
\rho_{15} = \sqrt{b_i^2 + \rho_{15}^2}, \quad \cos \theta_{15} = \frac{b_i}{\rho_{15}}, \quad \sin \theta_{15} = \frac{-\rho_7}{\rho_{15}}, \quad (253)
\]

\[
\rho_{31} = \sqrt{b_i^2 + \rho_{15}^2}, \quad \cos \theta_{31} = \frac{b_i}{\rho_{31}}, \quad \sin \theta_{31} = \frac{\rho_{15}}{\rho_{31}}. \quad (254)
\]

**C Appendix: Compiling Banded Oracle**

Suppose \( \vec{x} \in \text{Bool} \) for \( \vec{b} \in \text{Bool}^\Lambda \), and \( N_{\text{lvs}} = 2^\Lambda \). Suppose \( \alpha \) is a qubit, and \( \vec{\xi} = (\Lambda - 1, \ldots, 1, 0) \) is a vector of \( \Lambda \) qubits, and all these \( \Lambda + 1 \) qubits are distinct. In this Appendix, we will show how to compile the oracle evolution operator

\[
U_O = \sigma_X(\alpha) \sum_{\vec{b} \in \text{Bool}^\Lambda} x_0 P_{\vec{b}}(\vec{\xi}), \quad (255)
\]

when \( \vec{x} = [x_0, x_1, \ldots, x_{N_{\text{lvs}}-1}]^T \) is banded. If we envision \( \vec{x} \) as a sequence of ones and zeros, then we will call a band of \( \vec{x} \), any subsequence of \( \vec{x} \) consisting of adjacent terms, all of which are one. It \( \vec{x} \) contains a fixed (\( \Lambda \) independent) number of bands, we will say it is banded.

We will consider first the case when \( \vec{x} \) has a single front band. By this we mean that the first \( N_1 \) terms of \( \vec{x} \) are one, and the rest are zero. Thus

\[
\vec{x} = [1, 1, \ldots, 1, 0, 0, \ldots, 0]^T. \quad (256)
\]

Define a binary vector \( \vec{b}_{\text{max}} \) and its corresponding set of binary vectors \( S(\vec{b}_{\text{max}}) \) by

\[
\vec{b}_{\text{max}} = \text{bin}(N_1 - 1), \quad S(\vec{b}_{\text{max}}) = \{ \vec{b} \in \text{Bool}^\Lambda : 0 \leq \text{dec}(\vec{b}) \leq \text{dec}(\vec{b}_{\text{max}}) \}. \quad (257)
\]

In the single front band case, \( U_O \) reduces to:

\[
U_O = \sigma_X(\alpha) \sum_{\vec{b} \in S(\vec{b}_{\text{max}})} P_{\vec{b}}(\vec{\xi}). \quad (258)
\]

\( U_O \), in the form given by Eq. (258), is already compiled. But the length of this compilation can be reduced significantly by reducing \( \sum_{\vec{b} \in S(\vec{b}_{\text{max}})} P_{\vec{b}} \). Let us consider an example first, before dealing with the general case. Suppose \( \Lambda = 8 \) and \( \vec{b}_{\text{max}} = (0110, 1101) = \text{bin}(109) \). Although \( \sum_{\vec{b} \in S(\text{bin}(109))} P_{\vec{b}} \) is a sum of 109 projection operators of the type \( P_{\vec{b}} \) where \( \vec{b} \in \text{Bool}^8 \), it can be expressed as a sum of just six simpler projection operators:
\[ \sum_{\vec{b} \in S(\text{bin}(109))} P_{\vec{b}} = \begin{cases} P_{00\ldots}, & (\text{sum from 0000,0000 to 0011,1111}) \\ +P_{010\ldots}, & (\text{sum from 0100,0000 to 0101,1111}) \\ +P_{0110,0}, & (\text{sum from 0110,0000 to 0110,0111}) \\ +P_{0110,10}, & (\text{sum from 0110,1000 to 0110,1011}) \\ +P_{0110,1100} & \\ +P_{0110,1101} \end{cases} \] (259)

Fig. 15 gives a circuit diagram for Eq. (258), assuming \( \text{dec}(\vec{b}_{\text{max}}) = 109 \), and with \( \sum_{\vec{b} \in S(\text{bin}(109))} P_{\vec{b}} \) expressed in the simplified form given by the right hand side of Eq. (259).

Figure 15: Circuit diagram for Eq. (258), when the first 109 components of \( \vec{x} \) equal one, and the final \( 2^8 - 109 = 256 - 109 = 147 \) components equal zero.

It’s also interesting to consider an example in which \( \text{dec}(\vec{b}_{\text{max}}) \) is an even number instead of an odd one. When \( \text{dec}(\vec{b}_{\text{max}}) = 108 \) instead of 109, we must remove the operator marked by an asterisk in Eq. (259), and the operator marked by an asterisk in Fig. 15.

The pattern of the control vertices in Eq. (259) and Fig. 15 is not hard to uncover. There is exactly one MCNOT for each nonzero bit in \( \text{bin}(109) \). In addition, there is one “final” MCNOT with controls equal to \( P_{\text{bin}(109)} \). Call \( \mathcal{K}(\beta) \) the “non-final” MCNOTs. There is precisely one of these for each bit \( \beta \in Z_{0,\Lambda} \) such that the \( \beta \) component of \( \vec{b}_{\text{max}} \) (i.e., \( (\vec{b}_{\text{max}})_\beta \)) equals one, and none when \( (\vec{b}_{\text{max}})_\beta = 0 \). \( \mathcal{K}(\beta) \) always has control \( P_0 \) at bit \( \beta \). At bits \( \alpha \in Z_{\beta+1,\Lambda-1} \), \( \mathcal{K}(\beta) \) has a control \( P_{(\vec{b}_{\text{max}})_\alpha} \). Hence, in general,
\[
\sum_{\vec{b} \in S(\vec{b}_{\text{max}})} P_{\vec{b}}(\xi) = \left\{ \sum_{\beta \in \mathbb{Z}_{0,\Lambda-1}} \delta((\vec{b}_{\text{max}})_{\beta}, 1) P_{\vec{b}}(\beta) \prod_{\alpha \in \mathbb{Z}_{\beta+1,\Lambda-1}} P_{\vec{b}_{\text{max}}}(\alpha) + P_{\vec{b}_{\text{max}}}(\xi) \right\}. \tag{260}
\]

Note that the number of MCNOTs in \(U_O\) is \(\mathcal{O}(\Lambda)\). The number of CNOTs in each of these MCNOTs is \(\mathcal{O}(\Lambda)\). Thus, the number of CNOTs in \(U_O\) is \(\mathcal{O}(\Lambda^2)\).

Now suppose \(\vec{x}\) has a single band which is, however, not at the front. Suppose it ranges from \(\text{dec}(\vec{b}_{\text{min}}) \neq 0\) to \(\text{dec}(\vec{b}_{\text{max}})\). Then just multiply \(U_O\) for a single front band up to \(\text{dec}(\vec{b}_{\text{min}}) - 1\) times \(U_O\) for a single front band up to \(\text{dec}(\vec{b}_{\text{max}})\). Multiple bands can be handled similarly.

Sometimes it is possible to apply \(\sigma_X\) on individual qubits and/or apply qubit permutations to \(U_O\) so as to get a new \(U_O\) with fewer bands. Fewer bands will lead to a shorter SEO of the type proposed in this appendix. \(\sigma_X\) on individual qubits and/or qubit permutations\(^8\) do not increase the length of a SEO if they are applied to the SEO on both sides, via a \(\odot\) product.

### D Appendix: Compiling Evolution Operators in Grover’s Algorithm

The goal of this appendix is not to say something new about Grover’s algorithm\(^{[27]}\). After all, Grover’s algorithm has been studied so extensively in the literature that it’s almost impossible to say anything new about it. The goal of this appendix is, rather, to review how one compiles the oracle and non-oracle evolution operators associated with Grover’s algorithm. This will allow the reader to compare the compilation of Grover’s algorithm with the compilation of FGG07 presented in this paper.

Let \(N_B\) be the number of qubits and \(N_S = 2^{N_B}\) the number of states used in Grover’s algorithm. Let \(\vec{x}_o \in \text{Bool}^{N_B}\) be the target state of the algorithm. Define

\[
|\mu\rangle = \frac{1}{\sqrt{N_S}} \sum_{\vec{x} \in \text{Bool}^{N_B}} |\vec{x}\rangle = \frac{1}{\sqrt{N_S}} [1, 1, \ldots, 1]^T, \tag{261}
\]

and

\[
\mu = |\mu\rangle \langle \mu| = \frac{1}{N_S} \begin{bmatrix} 1 & 1 & \ldots & 1 \\ 1 & 1 & \ldots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \ldots & 1 \\ 1 & 1 & \ldots & 1 \end{bmatrix}. \tag{262}
\]

Thus, in matrix notation, \(|\mu\rangle\) is an \(N_S\) dimensional column vector, and \(\mu\) is an \(N_S \times N_S\) matrix.

\(^8\)Don’t confuse qubit permutations with state permutations.
In Grover’s algorithm, one alternates between an oracle evolution operator, call it $U_{\text{corr}}$, and a non-oracle evolution operator, call it $U_{\text{bulk}}$. The oracle evolution operator depends on the target state whereas the non-oracle evolution operator does not. More precisely, these two evolution operators are defined by:

$$U_{\text{corr}} = (-1)^{|\vec{x}_o\rangle\langle\vec{x}_o|} = (-1)^{P_{\vec{x}_o}} = e^{i\pi P_{\vec{x}_o}}, \quad (263)$$

and

$$U_{\text{bulk}} = 2|\mu\rangle\langle\mu| - 1 = -(-1)^{|\mu\rangle\langle\mu|} = e^{i\pi(\mu+1)}. \quad (264)$$

In Grover’s algorithm, one applies $\sqrt{N_S}$ times the product $U_{\text{corr}} U_{\text{bulk}}$. A variant of this would be to apply $N_T = N_S^{\frac{1}{2} + \delta}$ times the product $U_{\text{corr} N_S^{\frac{1}{2} + \delta}} U_{\text{bulk} N_S^{\frac{1}{2} + \delta}}$, for some $\delta > 0$. One can think of this variant of Grover’s algorithm as a Trotterized Lie approximation:

$$e^{i(H_{\text{corr}} + H_{\text{bulk}})} \approx \left( e^{i \frac{H_{\text{corr}}}{N_S^{\frac{1}{2} + \delta}}} e^{i \frac{H_{\text{bulk}}}{N_S^{\frac{1}{2} + \delta}}} \right)^{N_S^{\frac{1}{2} + \delta}}, \quad (265)$$

for some $t \in \mathbb{R}$, where the Hamiltonians $H_{\text{corr}}$ and $H_{\text{bulk}}$ are given by

$$\frac{i}{N_S^{\frac{1}{2} + \delta}} U_{\text{corr}}^{N_S^{\frac{1}{2} + \delta}} \Rightarrow \frac{H_{\text{corr}}}{N_S^{\frac{1}{2} + \delta}} = \frac{\pi P_{\vec{x}_o}}{N_S^{\frac{1}{2} + \delta}} \quad (266a)$$

and

$$\frac{i}{N_S^{\frac{1}{2} + \delta}} U_{\text{bulk}}^{N_S^{\frac{1}{2} + \delta}} \Rightarrow \frac{H_{\text{bulk}}}{N_S^{\frac{1}{2} + \delta}} = \frac{\pi (1 + \mu)}{N_S^{\frac{1}{2} + \delta}}. \quad (266b)$$

This variant of Grover’s algorithm is a quantum walk over a fully connected graph with $N_S$ nodes. Transitions occur on this graph along edges connecting distinct nodes and also from a node back to itself. Self transitions occur with strength proportional to $1 + \frac{1}{N_S}$ for all nodes except the one representing the target state. Self transitions for the target node occur with strength proportional to $2 + \frac{1}{N_S}$, larger than the strength for the other self transitions. Transitions along the edges connecting distinct nodes occur with vanishing strength proportional to $\frac{1}{N_S}$.

Compiling the oracle evolution operator for Grover’s algorithm is trivial. Let $\tilde{\xi}$ denote the $N_B$-dimensional vector of qubit positions for the $N_B$ primary qubits used in Grover’s algorithm, and let $\alpha$ denote the qubit position of an additional (not in $\tilde{\xi}$) ancilla qubit. Then

$$U_{\text{corr}} = (-1)^{P_{\vec{x}_o}(\tilde{\xi})} = (-1)^{(\alpha)P_{\vec{x}_o}(\tilde{\xi})} |1\rangle_\alpha \quad (267a)$$

$$= \sigma_Z(\alpha)|P_{\vec{x}_o}(\tilde{\xi})|1\rangle_\alpha \quad (267b)$$

$$= H(\alpha)\sigma_X(\alpha)|P_{\vec{x}_o}(\tilde{\xi})\rangle(H(\sigma_X))|\alpha\rangle_\alpha. \quad (267c)$$

We used $\sigma_Z = (-1)^n$, $\sigma_X|0\rangle = |1\rangle$ and $H \sigma_X H = \sigma_Z$. 

}\end{document}
Compiling the non-oracle evolution operator for Grover’s algorithm is also trivial. One notes that

\[ |\mu\rangle = H^{\otimes N_B}|0\rangle , \]  

(268)
since the first column of \( H^{\otimes N_B} \) is all ones. Therefore,

\[ \mu = H^{\otimes N_B} |0\rangle \langle 0| H^{\otimes N_B} . \]  

(269)

Eq. (269) is merely the eigenvalue decomposition (and SVD) of \( \mu \). Since \( \mu \) is a circulant matrix, we could have obtained Eq. (269) from the eigenvalue decomposition of circulant matrices presented in Section 3.1. An immediate consequence of Eq. (269) is

\[ U_{\text{bulk}} = -H^{\otimes N_B} (-1)^{P_0,0,...,0} H^{\otimes N_B} = -H^{\otimes N_B} (-1)^{P_0,0,...,0} H^{\otimes N_B} . \]  

(270)

One can compile \((-1)^{P_0,0,...,0}\) in the same way that \((-1)^{P_{\vec{x}_0}}\) was compiled above.

Note that we have expressed both \( U_{\text{bulk}} \) and \( U_{\text{corr}} \) as SEOs containing a single MCNOT. This MCNOT can be expressed as a SEO with \( \mathcal{O}(N_B) \) CNOTs.

References

[1] E. Farhi, J. Goldstone, S. Gutmann, “A Quantum Algorithm for the Hamiltonian NAND Tree”, arXiv:quant-ph/0702144

[2] A. Childs, R. Cleve, S. Jordan, D. Yeung, “Discrete-query quantum algorithm for NAND trees”, arXiv:quant-ph/0702160

[3] A. Childs, B. Reichardt, R. Spalek, S. Zhang, “Every NAND formula on N variables can be evaluated in time \( \mathcal{O}(N^{1/2+\epsilon}) \)”, arXiv:quant-ph/0703015

[4] A. Ambainis, “A nearly optimal discrete query quantum algorithm for evaluating NAND formulas”, arXiv:0704.3628

[5] A. Childs, R. Cleve, E. Deotto, E. Farhi, S. Gutmann, D. Spielman, “Exponential algorithmic speedup by quantum walk”, arXiv:quant-ph/0209131

[6] D. Berry, G. Ahokas, R. Cleve, B. Sanders, “Efficient quantum algorithms for simulating sparse Hamiltonians”, arXiv:quant-ph/0508139

[7] A. Childs, “Quantum information processing in continuous time” (MIT 2004 Ph.D. thesis)

[8] Andrew P. Hines, P.C.E. Stamp, “Quantum Walks, Quantum Gates and Quantum Computers”, arXiv:quant-ph/0701088
[9] G.H. Golub and C.F. Van Loan, *Matrix Computations, Third Edition* (John Hopkins Univ. Press, 1996).

[10] R.R. Tucci, “QC Paulinesia”, quant-ph/0407215

[11] R. M. Gray, “Toeplitz and Circulant Matrices: A Review”, available at http://www-ee.stanford.edu/~gray/toeplitz.pdf

[12] N. Hatano, M. Suzuki, “Finding Exponential Product Formulas of Higher Orders”, arXiv:math-ph/0506007

[13] C. Zachos, “Crib Notes on Campbell-Baker-Hausdorff expansions”, http://www.hep.anl.gov/~czachos/CBH.pdf

[14] M. Reinsch, “A Simple Expression for the Terms in the Baker-Campbell-Hausdorff Series”, math-ph/9906007

[15] Barenco et al., “Elementary Gates for Quantum Computation”, arXiv:quant-ph/9503016

[16] R. R. Tucci, “Simplifying Quantum Circuits via Circuit Invariants and Dressed CNOTs”, arXiv:quant-ph/0606061

[17] R.R. Tucci, “A Rudimentary Quantum Compiler(2nd Ed.)”, arXiv:quant-ph/9902062

[18] V. Bergholm, J. Vartiainen, M. Mottonen, M. Salomaa, “Quantum circuit for a direct sum of two-dimensional unitary operators”, quant-ph/0410066

[19] www.ar-tiste.com/qubiter.htm

[20] V. Shende, S. Bullock, I. Markov, “A Practical Top-down Approach to Quantum Circuit Synthesis”, quant-ph/0406176

[21] R.R. Tucci, “Qubiter Algorithm Modification, Expressing Unstructured Unitary Matrices with Fewer CNOTs”, arXiv:quant-ph/0411027

[22] G. Vidal, C.M. Dawson, “A Universal Quantum Circuit for Two-qubit Transformations with 3 CNOT Gates”, quant-ph/0307177

[23] V.V. Shende, I.L. Markov, S.S. Bullock, “On Universal Gate Libraries and Generic Minimal Two-qubit Quantum Circuits”, quant-ph/0308033

[24] R.R. Tucci, “Quantum Fast Fourier Transform Viewed as a Special Case of Recursive Application of Cosine-Sine Decomposition”, arXiv:quant-ph/0411097

[25] Y. Nakajima, Y. Kawano, H. Sekigawa, “A new algorithm for producing quantum circuits using KAK decompositions”, arXiv:quant-ph/0509196

63
[26] Don Coppersmith, “An approximate Fourier transform useful in quantum factoring”, (1994 IBM Internal Report), quant-ph/0201067

[27] See http://en.wikipedia.org/wiki/Grover’s_algorithm