Minimal-memory realization of pearl-necklace encoders of general quantum convolutional codes

Monireh Houshmand and Saied Hosseini-Khayat
Department of Electrical Engineering,
Ferdowsi University of Mashhad, Iran
{monirehhoushmand, saied.hosseini}@gmail.com

September 14, 2010

Abstract

Quantum convolutional codes, like their classical counterparts, promise to offer higher error correction performance than block codes of equivalent encoding complexity, and are expected to find important applications in reliable quantum communication where a continuous stream of qubits is transmitted. Grassl and Roetteler devised an algorithm to encode a quantum convolutional code with a "pearl-necklace encoder." Despite their theoretical significance as a neat way of representing quantum convolutional codes, they are not well-suited to practical realization. In fact, there is no straightforward way to implement any given pearl-necklace structure. This paper closes the gap between theoretical representation and practical implementation. In our previous work, we presented an efficient algorithm for finding a minimal-memory realization of a pearl-necklace encoder for Calderbank-Shor-Steane (CSS) convolutional codes. This work extends our previous work and presents an algorithm for turning a pearl-necklace encoder for a general (non-CSS) quantum convolutional code into a realizable quantum convolutional encoder. We show that a minimal-memory realization depends on the commutativity relations between the gate strings in the pearl-necklace encoder. We find a realization by means of a weighted graph which details the non-commutative paths through the pearl-necklace. The weight of the longest path in this graph is equal to the minimal amount of memory needed to implement the encoder. The algorithm has a polynomial-time complexity in the number of gate strings in the pearl-necklace encoder.

Quantum error correction codes are used to protect quantum information from decoherence and operational errors [2, 3, 4, 5, 6, 7, 8, 9, 10, 11]. Depending on their approach to error control, error correcting codes can be divided into two general classes: block codes and convolutional codes. In the case of a block code, the original state is first divided into a finite number of blocks of fixed length. Each block is then encoded separately and the encoding is independent of the other blocks. On the other hand, a quantum convolutional code [12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25] encodes an incoming stream of quantum information into an outgoing stream. Fast decoding algorithms exist for quantum convolutional codes [26] and in general, they are preferable in terms of their performance-complexity tradeoff [18].

The encoder for a quantum convolutional code has a representation as a *convolutional encoder* or as a *pearl-necklace* encoder. The convolutional encoder [12, 13], [26] consists of a single unitary repeatedly applied to a stream of quantum data (see Figure 1(a)). On the other hand, the pearl-necklace encoder (see Figure 1(b)) consists of several strings of the same unitary applied to the quantum data stream. Grassl and Rötteler [14] proposed an algorithm for encoding any quantum convolutional code with a pearl-necklace

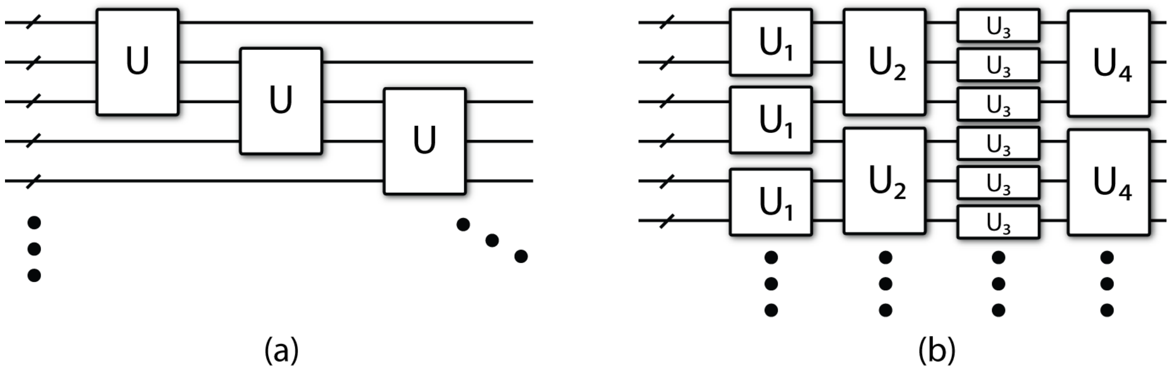

Figure 1: Two different representations of the encoder for a quantum convolutional code. (a) Representation of the encoder as a convolutional encoder. (b) Representation of the encoder as a pearl-necklace encoder [1].

encoders. The algorithm consists of a sequence of elementary encoding operations. Each of these elementary encoding operations corresponds to a gate string in the pearl-necklace encoder.

The amount of required memory plays a key role for implementation of any encoder, since this amount will result in overhead in the implementation of communication protocols. Hence any reduction in the required amount of memory will help in practical implementation of quantum computer.

It is trivial to determine the amount of memory required for implementation of a convolutional encoder: it is equal to the number of qubits that are fed back into the next iteration of the unitary that acts on the stream. For example, the convolutional encoders in the Figures 1(a), 2(c) and 4(b) require two, one and four frames of memory qubits, respectively.

In contrast, the practical realization of a pearl-necklace encoder is not explicitly clear. To make it realizable, one should rearrange the gate strings in the pearl-necklace encoder so that it becomes a convolutional encoder. In [1] we proposed an algorithm for finding the minimal-memory realization of a pearl necklace encoder for the CSS class of convolutional codes. This kind of encoder consists of CNOT gate strings only [27].

In this paper we extend our work to find the minimal-memory realization of a pearl-necklace encoder for a general (non-CSS) convolutional code. A general case includes all gate strings that are in the shift-invariant Clifford group [14]: Hadamard gates, phase gates, controlled-phase gate string, finite-depth and infinite-depth [23, 25] CNOT operations. We show that there are many realization for a given pearl-necklace encoder which are obtained considering non-commutativity relations of gate strings in the pearl-necklace encoder. Then for finding the minimal-memory realization a specific graph, called non-commutativity graph is introduced. Each vertex in the non-commutativity graph, corresponds to a gate string in the pearl-necklace encoder. The graph features a directed edge from one vertex to another if the two corresponding gate strings do not commute. The weight of a directed edge depends on the degrees of the two corresponding gate strings and their type of non-commutativity. The weight of the longest path in the graph is equal to the minimal memory requirement for the pearl-necklace encoder. The complexity for constructing this graph is quadratic in the number of gate strings in the encoder.

The paper is organized as follows. In Section 1, we introduce some definitions and notation that are used in the rest of paper. In Section 2, we define three different types of non-commutativity and then propose an algorithm to find the minimal memory requirements in a general case. In Section 3, we will summarize the contribution of this paper.

1 Definitions and notation

We first provide some definitions and notation which are useful for our analysis later on. The gate strings in the pearl-necklace encoder and the gates in the convolutional encoder are numbered from left to right. We denote the i^{th} gate string in the pearl-necklace encoder, $\overline{U}_i$, and the i^{th} gate in the convolutional encoder, U_i.

Let $\overline{U}$, without any index specified, denote a particular infinitely repeated sequence of U gates, where the sequence contains the same U gate for every frame of qubits.

Let U be either CNOT or CPHASE gate. The notation $\overline{U}\left(a, bD^l\right)$ refers to a string of gates in a pearl-necklace encoder and denotes an infinitely repeated sequence of U gates from qubit a to qubit b in every frame where qubit b is in a frame delayed by l. [1]

Let U be either phase or Hadamard gate. The notation $\overline{U}\,(b)$ refers to a string of gates in a pearl-necklace encoder and denotes an infinitely repeated sequence of U gates which act on qubit b in every frame. By convention we call this qubit, the target of $\overline{U}\,(b)$ during this paper.

If $\overline{U_i}$ is $\overline{\text{CNOT}}$ or $\overline{\text{CPHASE}}$ the notation a_i, b_i, and l_i are used to denote its source index, target index and degree, respectively. If $\overline{U_i}$ is $\overline{H}$ or $\overline{P}$ the notation b_i is used to denote its target index.

For example, the strings of gates in Figure 2(a) correspond to:

$$\overline{H}\,(3)\,\overline{\text{CPHASE}}\,(1, 2D)\,\overline{\text{CNOT}}\,(1, 3)\,, \tag{1}$$

$b_1 = 1$, $a_2 = 1$, $b_2 = 2$, $l_2 = 1$, $a_3 = 1$, $b_3 = 3$, and $l_3 = 0$.

Suppose the number of gate strings in the pearl-necklace encoder is N. The members of the sets I^+_{CNOT}, I^-_{CNOT}, I^+_{CPHASE}, and I^-_{CPHASE} are the indices of gate strings in the encoder which are $\overline{\text{CNOT}}$ with non-negative degree, $\overline{\text{CNOT}}$ with negative degree, $\overline{\text{CPHASE}}$ with non-negative degree and $\overline{\text{CPHASE}}$ with negative degree respectively:

$$I^+_{\text{CNOT}} = \{i|\,\overline{U}_i \text{ is } \overline{\text{CNOT}}, l_i \geq 0, i \in \{1, 2, \cdots, N\}\},$$

$$I^-_{\text{CNOT}} = \{i|\,\overline{U}_i \text{ is } \overline{\text{CNOT}}, l_i < 0, i \in \{1, 2, \cdots, N\}\},$$

$$I^+_{\text{CPHASE}} = \{i|\,\overline{U}_i \text{ is } \overline{\text{CPHASE}}, l_i \geq 0, i \in \{1, 2, \cdots, N\}\},$$

$$I^-_{\text{CPHASE}} = \{i|\,\overline{U}_i \text{ is } \overline{\text{CPHASE}}, l_i < 0, i \in \{1, 2, \cdots, N\}\}.$$

The members of the sets I_{H} and I_{P} are the indices of gate strings of the encoder which are $\overline{H}$ and $\overline{P}$ respectively:

$$I_{\text{H}} = \{i|\,\overline{U}_i \text{ is } \overline{H}, i \in \{1, 2, \cdots, N\}\},$$

$$I_{\text{P}} = \{i|\,\overline{U}_i \text{ is } \overline{P}, i \in \{1, 2, \cdots, N\}\}.$$

Our convention for numbering the frames upon which the unitary of a convolutional encoder acts is from "bottom" to "top." Figure 5(b) illustrates this convention for a convolutional encoder. If U_i is CNOT or CPHASE gate, then let σ_i and τ_i denote the frame index of the respective source and target qubits of the U_i gate in a convolutional encoder. If U_i is Hadamard or Phase gate, let τ_i denote the frame index of the target qubit of the U_i gate in a convolutional encoder. For example, consider the convolutional encoder in Figure 5(b). The convolutional encoder in this figure consists of six gates; $\tau_1 = 0$, $\tau_2 = 0$, $\sigma_3 = 0$, $\tau_3 = 1$, $\sigma_4 = 2$, $\tau_4 = 0$, $\sigma_5 = 3$, $\tau_5 = 2$, $\sigma_6 = 4$, and $\tau_6 = 3$.

While referring to a convolutional encoder, the following notation are defined as follows:
The notation $\text{CNOT}(a, b)\,(\sigma, \tau)$ denotes a CNOT gate from qubit a in frame σ to qubit b in frame τ.

[1] Instead of the previously used notation $\overline{U}(a, b)(D^l)$, we prefer to use $\overline{U}\left(a, bD^l\right)$ as it seems to better represent the concept.

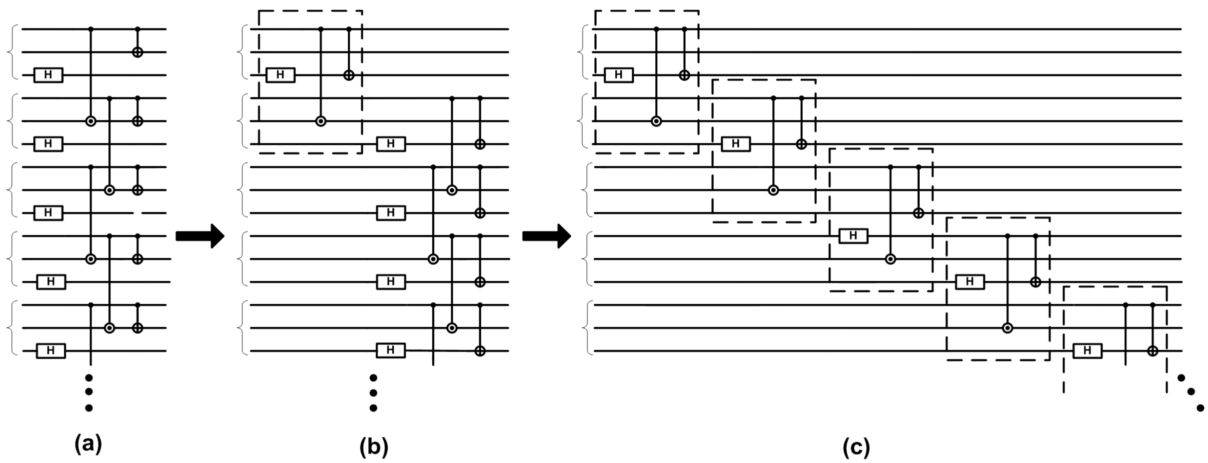

Figure 2: Simple (since all gate strings commute with each other) example of the rearrangement of a pearl-necklace encoder for a non-CSS code into a convolutional encoder. (a) The pearl-necklace encoder consists of the gate strings $\overline{H}\,(1)\,\overline{\text{CPHASE}}\,(1,2)\,(D)\,\overline{\text{CNOT}}\,(1,3)$. (b) The rearrangement of gates after the first three by shifting them to the right. (c) The repeated application of the procedure in (b) realizes a convolutional encoder from a pearl-necklace encoder.

The notation CPHASE$(a, b)\,(\sigma, \tau)$ denotes a CPHASE gate from qubit a in frame σ to qubit b in frame τ.
The notation $H(b)(\tau)$ denotes a Hadamard gate which acts on qubit b in frame τ.
The notation $P(b)(\tau)$ denotes a Phase gate which acts on qubit b in frame τ.

For example the gates in Figure 5(b) correspond to:

$$H\,(1)\,(0)P\,(1)\,(0)\text{CPHASE}\,(1,2)\,(0,1)\,\text{CPHASE}\,(2,3)\,(2,0)\ \ \text{CNOT}\,(3,2)\,(3,2)\,\text{CNOT}\,(2,3)\,(4,3)\,.$$

2 Memory requirements for an arbitrary pearl-necklace encoder

As discussed before, for finding the practical realization of a pearl-necklace encoder it is required to rearrange the gates as a convolutional encoder.

To do this rearrangement, we must first find a set of gates consisting of a single gate for each gate string in the pearl-necklace encoder such that all the gates that remain after the set commute with it. Then we can shift all these gates to the right and infinitely repeat this operation on the remaining gates to obtain a convolutional encoder. When all gates in the pearl-necklace encoder commute with each other, there is no constraint on frame indices of target (source) qubits of gates in the convolutional encoder [1]. (Figure 2 shows an example of the rearrangement of commuting gate strings into a convolutional encoder.) On the other hand, when the gate strings do not commute, the constraint of commutativity of the remaining gates with the chosen set results in constraints on frame indices of target (source) qubits of gates in the convolutional encoder.

In the following sections, after defining different types of non-commutativity and their imposed constraints, the algorithm for finding the minimal-memory convolutional encoder for an arbitrary pearl-necklace encoder is presented.

2.1 Different types of non-commutativity and their imposed constraints

There may arise three types of non-commutativity for any two gate strings of shift-invariant Clifford: source-target non-commutativity, target-source non-commutativity and target-target non-commutativity. Each imposes a different constraint on frame indices of gates in the convolutional encoder. These types of non-commutativity and their constraints are explained in the following sections.

2.1.1 Source-target non-commutativity

The gate strings in (2-5) below do not commute with each other. In all of them, the index of each source qubit in the first gate string is the same as the index of each target qubit in the second gate string, therefore we call this type of non-commutativity *source-target non-commutativity.*

$$\overline{\text{CNOT}}(a, bD^l)\overline{\text{CNOT}}(a', b'D^{l'}), \text{ where } a = b', \tag{2}$$

$$\overline{\text{CPHASE}}(a, bD^l)\overline{\text{CNOT}}(a', b'D^{l'}), \text{ where } a = b', \tag{3}$$

$$\overline{\text{CNOT}}(a, bD^l)\overline{H}(b'), \text{ where } a = b', \tag{4}$$

$$\overline{\text{CPHASE}}(a, bD^l)\overline{H}(b'), \text{ where } a = b'. \tag{5}$$

With an analysis similar to the analysis in Section 3.1 of [1], it can be proved that the following inequality applies to any correct choice of a convolutional encoder that implements either of the transformations in (2-5):

$$\sigma \leq \tau', \tag{6}$$

where σ and τ' denote the frame index of the source qubit of the first gate and the frame index of the target qubit of the second gate in a convolutional encoder respectively. We call the inequality in (6), *source-target constraint.*

As an example, the gate strings of the pearl-necklace encoder, $\overline{\text{CPHASE}}(2, 3D)\overline{\text{CNOT}}(1, 2D)$, (Figure 3(a)) have source-target non-commutativity. A correct choice of convolutional encoder is (the encoder depicted over a first arrow in the Figure 3):

$$\text{CPHASE}(2, 3)(1, 0)\text{CNOT}(1, 2)(2, 1). \tag{7}$$

In this case $\sigma = 1 \leq \tau' = 1$. Since the source-target constraint is satisfied the remaining gates after the chosen set in Figure3(b) can be shifted to the right. Repeated application of the procedure in (b) realizes a convolutional encoder representation from a pearl-necklace encoder(Figure3(c)).

The following Boolean function is used to determine whether this type of non-commutativity exists for two gate strings:

$$\text{Source-Target}\left(\overline{U}_i, \overline{U}_j\right).$$

This function takes two gate strings $\overline{U}_i$ and $\overline{U}_j$ as input. It returns TRUE if $\overline{U}_i$ and $\overline{U}_j$ have source-target non-commutativity and returns FALSE otherwise.

2.1.2 Target-source non-commutativity

It is obvious that the gate strings in (8-11) do not commute. In all of them, the index of each target qubit in the first gate string is the same as the index of each source qubit in the second gate string. Therefore we call this type of non-commutativity, *target-source non-commutativity.*

$$\overline{\text{CNOT}}(a, bD^l)\overline{\text{CNOT}}(a', b'D^{l'}), \text{ where } b = a', \tag{8}$$

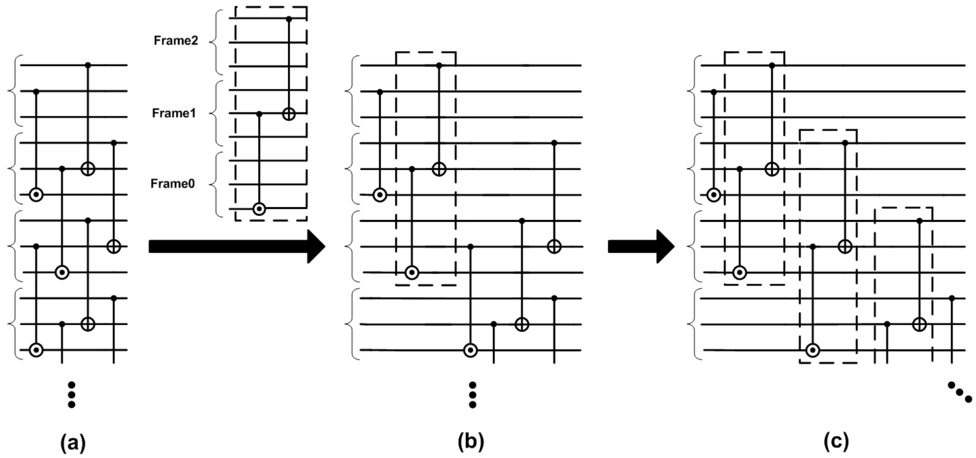

Figure 3: Finding a correct choice for a two non-commutative gate strings . (a) The pearl-necklace encoder consists of the gate strings $\overline{\text{CPHASE}}(2,3D)\overline{\text{CNOT}}(1,2D)$, which have source-target non-commutativity. (b) The rearrangement of gates after the first three by shifting them to the right. (c) The repeated application of the procedure in (b) realizes a convolutional encoder from a pearl-necklace encoder.

$$\overline{\text{CNOT}}(a,bD^{l})\overline{\text{CPHASE}}(a',b'D^{l'}),\ \text{where } b=a', \tag{9}$$

$$\overline{H}(b)\overline{\text{CNOT}}(a',b'D^{l'}),\ \text{where } b=a', \tag{10}$$

$$\overline{H}(b)\overline{\text{CPHASE}}(a',b'D^{l'}),\ \text{where } b=a'. \tag{11}$$

With an analysis similar to the analysis in Section 3.1 of [1], it can be proved that the following inequality applies to any correct choice of a convolutional encoder that implements either of the transformations in (8-11):

$$\tau \leq \sigma', \tag{12}$$

where τ and σ' denote the frame index of the target qubit of the first gate and the frame index of the source qubit of the second gate in a convolutional encoder respectively. We call the inequality in (12), *target-source constraint.*

The following Boolean function is used to determine whether target-source non-commutativity exists for two gate strings:

$$\text{Target-Source}\left(\overline{U}_i,\overline{U}_j\right).$$

This function takes two gate strings $\overline{U}_i$ and $\overline{U}_j$ as input. It returns TRUE if $\overline{U}_i$ and $\overline{U}_j$ have target-source non-commutativity and returns FALSE otherwise.

2.1.3 Target-target non-commutativity

It is obvious that the gate strings in (13-22) do not commute. In all of them, the index of each target qubit in the first gate string is the same as the index of each target qubit in the second gate string. Therefore we call this type of non-commutativity, *target-target non-commutativity.*

$$\overline{\text{CPHASE}}(a,bD^{l})\ \overline{\text{CNOT}}(a',b'D^{l'}),\ \text{where } b=b', \tag{13}$$

$$\overline{\text{CNOT}}(a,bD^{l})\overline{\text{CPHASE}}(a',b'D^{l'}),\ \text{where } b=b', \tag{14}$$

$$\overline{\text{CNOT}}(a, bD^l)\overline{H}(b),\ \text{where } b = b', \tag{15}$$

$$\overline{\text{CPHASE}}(a, bD^l)\overline{H}(b'),\ \text{where } b = b', \tag{16}$$

$$\overline{H}(b)\overline{\text{CNOT}}(a', b'D^{l'}),\ \text{where } b = b', \tag{17}$$

$$\overline{H}(b)\overline{\text{CPHASE}}(a', b'D^{l'}),\ \text{where } b = b', \tag{18}$$

$$\overline{\text{CNOT}}(a, bD^l)\overline{P}(b'),\ \text{where } b = b', \tag{19}$$

$$\overline{P}(b)\overline{\text{CNOT}}(a', b'D^{l'}),\ \text{where } b = b', \tag{20}$$

$$\overline{P}(b)\overline{H}(b'),\ \text{where } b = b', \tag{21}$$

$$\overline{H}(b)\overline{P}(b'),\ \text{where } b = b'. \tag{22}$$

With a analysis similar to the analysis in Section 3.1 of [1], it can be proved that the following inequality applies to any correct choice of a convolutional encoder that implements either of the transformations in (13-22):

$$\tau \le \tau', \tag{23}$$

where τ and τ' denote the frame index of the target qubit of the first gate and the frame index of the target qubit of the second gate in a convolutional encoder respectively. We call the inequality in (23), *target-target constraint.* The following Boolean function is used to determine whether target-target non-commutativity exists for two gate strings:

$$\text{Target-Target}\left(\overline{U}_i, \overline{U}_j\right) = \text{TRUE}.$$

This function takes two gate strings $\overline{U}_i$ and $\overline{U}_j$ as input. It returns TRUE if $\overline{U}_i$ and $\overline{U}_j$ have target-target non-commutativity and returns FALSE otherwise.

Consider the j^{th} gate string, $\overline{U}_j$ in the encoder. It is important to consider the gate strings preceding this one that do not commute with this gate string and categorize them based on the type of non-commutativity. Therefore we define the following sets:

$$(\mathcal{S}-\mathcal{T})_j = \{i \mid \text{Source-Target}(\overline{U}_i, \overline{U}_j) = \text{TRUE}, i \in \{1, 2, \cdots, j-1\}\},$$

$$(\mathcal{T}-\mathcal{S})_j = \{i \mid \text{Target-Source}(\overline{U}_i, \overline{U}_j) = \text{TRUE}, i \in \{1, 2, \cdots, j-1\}\},$$

$$(\mathcal{T}-\mathcal{T})_j = \{i \mid \text{Target-Target}(\overline{U}_i, \overline{U}_j) = \text{TRUE}, i \in \{1, 2, \cdots, j-1\}\}.$$

2.2 The proposed algorithm for finding minimal memory requirements for an arbitrary pearl-necklace encoder

In this section we find the minimal-memory realization for an arbitrary pearl-necklace encoder which include all gate strings that are in shift-invariant Clifford group: Hadamard gates, Phase gates, controlled-phase and finite depth and infinite-depth controlled-NOT gate strings. To achieve this goal, we consider any non-commutativity that may exist for a particular gate and its preceding gates. Suppose that a pearl-necklace encoder features the following succession of N gate strings:

$$\overline{U_1}, \overline{U_2}, \cdots, \overline{U_N}. \tag{24}$$

If the first gate string is $\overline{\text{CNOT}}(a_1, b_1D^{l_1}), l_1 \ge 0$, the first gate in the convolutional encoder is

$$\text{CNOT}(a_1, b_1)(\sigma_1 = l_1, \tau_1 = 0). \tag{25}$$

If the first gate string is $\overline{\text{CNOT}}(a_1, b_1 D^{l_1}), l_1 < 0$ the first gate in the convolutional encoder is

$$\text{CNOT}(a_1, b_1)(\sigma_1 = 0, \tau_1 = |l_1|). \tag{26}$$

If the first gate string is $\overline{\text{CPHASE}}(a_1, b_1 D^{l_1}), l_1 \geq 0$ the first gate in the convolutional encoder is

$$\text{CPHASE}(a_1, b_1)(\sigma_1 = l_1, \tau_1 = 0). \tag{27}$$

If the first gate string is $\overline{\text{CPHASE}}(a_1, b_1 D^{l_1}), l_1 < 0$ the first gate in the convolutional encoder is

$$\text{CPHASE}(a_1, b_1)(\sigma_1 = 0, \tau_1 = |l_1|). \tag{28}$$

If the first gate string is $\overline{H}(b_1)$ or $\overline{P}(b_1)$ the first gate in the convolutional encoder is as follows respectively:

$$H(b_1)(0), \tag{29}$$

$$P(b_1)(0). \tag{30}$$

For the target indices of each gate j where $2 \leq j \leq N$, we should choose a value for τ_j that satisfies all the constraints that the gates preceding it impose.

First consider $\overline{U_j}$ is the $\overline{\text{CNOT}}$ or $\overline{\text{CPHASE}}$ gate, then the following inequalities must be satisfied to target index of $\overline{U_j}$, τ_j:

By applying the source-target constraint in (6) we have:

$$\begin{aligned} \sigma_i \leq \tau_j \quad \forall i \in (\mathcal{S} - \mathcal{T})_j, \\ \therefore\ \tau_i + l_i \leq \tau_j \quad \forall i \in (\mathcal{S} - \mathcal{T})_j, \\ \therefore\ \max\{\tau_i + l_i\}_{i \in (\mathcal{S}-\mathcal{T})_j} \leq \tau_j, \end{aligned} \tag{31}$$

by applying the target-source constraint in (12) we have:

$$\begin{aligned} \tau_i \leq \sigma_j \quad \forall i \in (\mathcal{T} - \mathcal{S})_j \\ \therefore\ \tau_i \leq \tau_j + l_j \quad \forall i \in (\mathcal{T} - \mathcal{S})_j, \\ \therefore\ \tau_i - l_j \leq \tau_j \quad \forall i \in (\mathcal{T} - \mathcal{S})_j, \\ \therefore\ \max\{\tau_i - l_j\}_{i \in (\mathcal{T}-\mathcal{S})_j} \leq \tau_j. \end{aligned} \tag{32}$$

By applying the target-target constraint in (23) we have:

$$\begin{aligned} \tau_i \leq \tau_j \quad \forall i \in (\mathcal{T} - \mathcal{T})_j \\ \therefore\ \max\{\tau_i\}_{i \in (\mathcal{T}-\mathcal{T})_j} \leq \tau_j. \end{aligned} \tag{33}$$

The following constraint applies to the frame index τ_j of the target qubit by applying (31-33):

$$\max\{\{\tau_i + l_i\}_{i \in (\mathcal{S}-\mathcal{T})_j}, \{\tau_i - l_j\}_{i \in (\mathcal{T}-\mathcal{S})_j}, \{\tau_i\}_{i \in (\mathcal{T}-\mathcal{T})_j}\} \leq \tau_j. \tag{34}$$

Thus, the minimal value for τ_j (which corresponds to the minimal-memory realization) that satisfies all the constraints is:

$$\tau_j = \max\{\{\tau_i + l_i\}_{i \in (\mathcal{S}-\mathcal{T})_j}, \{\tau_i - l_j\}_{i \in (\mathcal{T}-\mathcal{S})_j}, \{\tau_i\}_{i \in (\mathcal{T}-\mathcal{T})_j}\}. \tag{35}$$

It can be easily shown that there is no constraint for the frame index τ_j if the gate string $\overline{U_j}$ commutes with all previous gate strings. Thus if $l_j \geq 0$ we choose the frame index τ_j as follows:

$$\tau_j = 0. \tag{36}$$

and if $l_j < 0$ we choose τ_j as follows:

$$\tau_j = |l_j|. \tag{37}$$

If $l_j \geq 0$, a good choice for the frame index τ_j, by considering (35) and (36) is as follows:

$$\tau_j = \max\{0, \{\tau_i + l_i\}_{i \in (\mathcal{S}-\mathcal{T})_j}, \{\tau_i - l_j\}_{i \in (\mathcal{T}-\mathcal{S})_j}, \{\tau_i\}_{i \in (\mathcal{T}-\mathcal{T})_j}\}. \tag{38}$$

and if $l_j < 0$, a good choice for the frame index τ_j, by considering (35) and (37) is as follows:

$$\tau_j = \max\{|l_j|, \{\tau_i + l_i\}_{i \in (\mathcal{S}-\mathcal{T})_j}, \{\tau_i - l_j\}_{i \in (\mathcal{T}-\mathcal{S})_j}, \{\tau_i\}_{i \in (\mathcal{T}-\mathcal{T})_j}\}. \tag{39}$$

Now consider $\overline{U_j}$ is the $\overline{H}$, then the following inequalities must be satisfied to target index of $\overline{U_j}$, τ_j:

By applying the source-target constraint in (6) we have:

$$\begin{aligned} \sigma_i \leq \tau_j \quad \forall i \in (\mathcal{S}-\mathcal{T})_j, \\ \therefore\ \tau_i + l_i \leq \tau_j \quad \forall i \in (\mathcal{S}-\mathcal{T})_j, \\ \therefore\ \max\{\tau_i + l_i\}_{i \in (\mathcal{S}-\mathcal{T})_j} \leq \tau_j. \end{aligned} \tag{40}$$

By applying target-target constraint in (23) we have:

$$\begin{aligned} \tau_i \leq \tau_j \quad \forall i \in (\mathcal{S}-\mathcal{T})_j \\ \therefore\ \max\{\tau_i\}_{i \in (\mathcal{T}-\mathcal{T})_j} \leq \tau_j, \end{aligned} \tag{41}$$

The following constraint applies to the frame index τ_j of the target qubit by applying (40) and (41):

$$\max\{\{\tau_i + l_i\}_{i \in (\mathcal{S}-\mathcal{T})_j}, \{\tau_i\}_{i \in (\mathcal{T}-\mathcal{T})_j}\} \leq \tau_j.$$

Thus, the minimal value for τ_j (which corresponds to the minimal-memory realization) that satisfies all the constraints is:

$$\tau_j = \max\{\{\tau_i + l_i\}_{i \in (\mathcal{S}-\mathcal{T})_j}, \{\tau_i\}_{i \in (\mathcal{T}-\mathcal{T})_j}\}. \tag{42}$$

It can be easily shown that there is no constraint for the frame index τ_j if the gate string $\overline{U_j}$ commutes with all previous gate strings. Thus, in this case, we choose the frame index τ_j as follows:

$$\tau_j = 0. \tag{43}$$

A good choice for the frame index τ_j, by considering (42) and (43) is as follows:

$$\tau_j = \max\{0, \{\tau_i + l_i\}_{i \in (\mathcal{S}-\mathcal{T})_j}, \{\tau_i\}_{i \in (\mathcal{T}-\mathcal{T})_j}\}. \tag{44}$$

Now consider $\overline{U_j}$ is the $\overline{P}$, then by applying the target-target constraint in (23), the following inequality must be satisfied to target index of $\overline{U_j}$, τ_j:

$$\begin{aligned} \tau_i \leq \tau_j \quad \forall i \in (\mathcal{T}-\mathcal{T})_j \\ \therefore\ \max\{\tau_i\}_{i \in (\mathcal{T}-\mathcal{T})_j} \leq \tau_j. \end{aligned}$$

Thus, the minimal value for τ_j (which corresponds to the minimal-memory realization) that satisfies all the constraints is:

$$\tau_j = \max\{\tau_i\}_{i\in(\mathcal{T}-\mathcal{T})_j}. \tag{45}$$

It can be easily shown that there is no constraint for the frame index τ_j if the gate string $\overline{U_j}$ commutes with all previous gate strings. Thus, in this case, we choose the frame index τ_j as follows:

$$\tau_j = 0. \tag{46}$$

A good choice for the frame index τ_j, by considering (45) and (46) is as follows:

$$\tau_j = \max\{0, \{\tau_i\}_{i\in(\mathcal{T}-\mathcal{T})_j}\}. \tag{47}$$

2.2.1 Construction of the non-commutativity graph

We introduce the notion of a *non-commutative* graph, $\mathcal{G}$ in order to find the best values for the target qubit frame indices. The graph is a weighted, directed acyclic graph constructed from the non-commutativity relations of the gate strings in (24). Algorithm 1 presents pseudo code for constructing the non-commutativity graph. $\mathcal{G}$ consists of N vertices, labeled $1, 2, \cdots, N$, where the j^{th} vertex corresponds to the j^{th} gate string $\overline{U}_j$. It also has two dummy vertices, named "START" and "END." DrawEdge(i, j, w) is a function that draws a directed edge from vertex i to vertex j with an edge weight equal to w.

2.2.2 The longest path gives the minimal memory requirements

Theorem 1 below states that the weight of the longest path from the START vertex to the END vertex is equal to the minimal memory required for a convolutional encoder implementation.

Theorem 1. *The weight w of the longest path from the START vertex to END vertex in the non-commutativity graph $\mathcal{G}$ is equal to the minimal memory L that the convolutional encoder requires.*

Proof. We first prove by induction that the weight w_j of the longest path from the START vertex to vertex j in the non-commutativity graph $\mathcal{G}$ is

$$w_j = \tau_j. \tag{48}$$

Based on the algorithm, a zero-weight edge connects the START vertex to the first vertex, if $1 \in (I^+_{\text{CNOT}} \cup I^+_{\text{CPHASE}} \cup I_H \cup I_P)$ and in this case based on (25), (27), (29) and (30), $\tau_1 = 0$ therefore $w_1 = \tau_1 = 0$. An edge with the weight equal to $|l_1|$ connects the START vertex to the first gate if $1 \in (I^-_{\text{CNOT}} \cup I^-_{\text{CPHASE}})$, and based on (26) and (28), $\tau_1 = |l_1|$ therefore $w_1 = \tau_1 = |l_1|$. Thus the base step holds for the target index of the first gate in a minimal-memory convolutional encoder. Now suppose the property holds for the target indices of the first k gates in the convolutional encoder:

$$w_j = \tau_j \quad \forall j : 1 \le j \le k.$$

Suppose we add a new gate string $\overline{U}_{k+1}$ to the pearl-necklace encoder, and Algorithm 1 then adds a new vertex $k+1$ to the graph $\mathcal{G}$. Suppose $(k+1) \in (I^+_{\text{CNOT}} \cup I^+_{\text{CPHASE}})$. The following edges are added to $\mathcal{G}$:

1. A zero-weight edge from the START vertex to vertex $k+1$.

2. An l_i-weight edge from each vertex $\{i\}_{i\in(S-T)_{k+1}}$ to vertex $k+1$.

3. A $-l_{k+1}$-weight edge from each vertex $\{i\}_{i\in(T-S)_{k+1}}$ to vertex $k+1$.

Algorithm 1 Algorithm for determining the non-commutativity graph $\mathcal{G}$ for general case

```
N ← Number of gate strings in the pearl-necklace encoder.
Draw a START vertex.
for j := 1 to N do
  Draw a vertex labeled j for the j^th gate string \overline{U}_j
  if j ∈ (I^-_CNOT ∪ I^-_CPHASE) then
    DrawEdge(START, j, |l_j|)
  else
    DrawEdge(START, j, 0)
  end if
  for i := 1 to j − 1 do
    if j ∈ (I^+_CNOT ∪ I^+_CPHASE ∪ I^-_CNOT ∪ I^-_CPHASE) then
      if i ∈ (S − T)_j then
        DrawEdge(i, j, l_i )
      end if
      if i ∈ (T − S)_j then
        DrawEdge(i, j, −l_j)
      end if
      if i ∈ (T − T)_j then
        DrawEdge(i, j, 0)
      end if
    else
      if j ∈ I_H then
        if i ∈ (S − T)_j then
          DrawEdge(i, j, l_i)
        end if
        if i ∈ (T − T)_j then
          DrawEdge(i, j, 0)
        end if
      else
        if i ∈ (T − T)_j then
          DrawEdge(i, j, 0)
        end if
      end if
    end if
  end for
end for
Draw an END vertex.
for j := 1 to N do
  if j ∈ (I^+_CNOT ∪ I^+_CPHASE) then
    DrawEdge(j,END, l_j)
  else
    DrawEdge(j,END, 0)
  end if
end for
```

4. A zero-weight edge from each vertex $\{i\}_{i\in(\mathcal{T}-\mathcal{T})_{k+1}}$ to vertex $k+1$.

5. An l_{k+1}-weight edge from vertex $k+1$ to the END vertex.

It is clear that the following relations hold:

$$\begin{aligned} w_{k+1} &= \max\{0, \{w_i + l_i\}_{i\in(\mathcal{S}-\mathcal{T})_{k+1}}, \{w_i - l_{k+1}\}_{i\in(\mathcal{T}-\mathcal{S})_{k+1}}, \{w_i\}_{i\in(\mathcal{T}-\mathcal{T})_{k+1}}\}, \\ &= \max\{0, \{\tau_i + l_i\}_{i\in(\mathcal{S}-\mathcal{T})_{k+1}}, \{\tau_i - l_{k+1}\}_{i\in(\mathcal{T}-\mathcal{S})_{k+1}}, \{\tau_i\}_{i\in(\mathcal{T}-\mathcal{T})_{k+1}}\}. \end{aligned} \tag{49}$$

By applying (38) and (49) we have:

$$w_{k+1} = \tau_{k+1}.$$

In a similar way we can show that if the U_{k+1} is any other gate string of Clifford shift-invariant:

$$w_{k+1} = \tau_{k+1}.$$

The proof of the theorem then follows by considering the following equalities:

$$\begin{aligned} w &= \max\{\max_{i\in(I^+_{\text{CNOT}}\cup I^+_{\text{CPHASE}}\cup I_H\cup I_P)}\{w_i + l_i\}, \max_{i\in(I^-_{\text{CNOT}}\cup I^-_{\text{CPHASE}})}\{w_i\}\} \\ &= \max\{\max_{i\in(I^+_{\text{CNOT}}\cup I^+_{\text{CPHASE}}\cup I_H\cup I_P)}\{\tau_i + l_i\}, \max_{i\in(I^-_{\text{CNOT}}\cup I^-_{\text{CPHASE}})}\{\tau_i\}\} \\ &= \max\{\max_{i\in(I^+_{\text{CNOT}}\cup I^+_{\text{CPHASE}}\cup I_H\cup I_P)}\{\sigma_i\}, \max_{i\in(I^-_{\text{CNOT}}\cup I^-_{\text{CPHASE}})}\{\tau_i\}\}. \end{aligned}$$

The first equality holds because the longest path in the graph is the maximum of the weight of the path to the i^{th} vertex summed with the weight of the edge to the END vertex. The second equality follows by applying (48). The final equality follows because $\sigma_i = \tau_i + l_i$. It is clear that

$$\max\{\max_{i\in(I^+_{\text{CNOT}}\cup I^+_{\text{CPHASE}}\cup I_H\cup I_P)}\{\sigma_i\}, \max_{i\in(I^-_{\text{CNOT}}\cup I^-_{\text{CPHASE}})}\{\tau_i\}\},$$

is equal to minimal required memory for a minimal-memory convolutional encoder, hence the theorem holds. □

The final task is to determine the longest path in $\mathcal{G}$. Finding the longest path in a graph, in general is an NP-complete problem, but in a weighted, directed acyclic graph requires linear time with dynamic programming [28]. The non-commutativity graph $\mathcal{G}$ is an acyclic graph because a directed edge connects each vertex only to vertices for which its corresponding gate comes later in the pearl-necklace encoder.

The running time for the construction of the graph is quadratic in the number of gate strings in the pearl-necklace encoder. Since in Algorithm 1, the instructions in the inner **for** loop requires constant time $O(1)$. The sum of iterations of the **if** instruction in the j^{th} iteration of the outer **for** loop is equal to $j-1$. Thus the running time $T(N)$ of Algorithm 1 is

$$T(N) = \sum_{i=1}^{N}\sum_{k=1}^{j-1} O(1) = O(N^2).$$

Example 1: Consider the following succession of gate strings in a pearl-necklace encoder(Figure 4(a)):

$$\overline{H}\,(1)\,P\,(1)\,\overline{\text{CPHASE}}\left(1, 2D^{-1}\right)\overline{\text{CPHASE}}\left(2, 3D^{2}\right)\overline{\text{CNOT}}\,(3, 2D)\,\overline{\text{CNOT}}\,(2, 3D)\,,$$

Figure 5(a) draws $\mathcal{G}$ for this pearl-necklace encoder, after running Algorithm. The longest path through the graph is

$$\text{START} \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow \text{END},$$

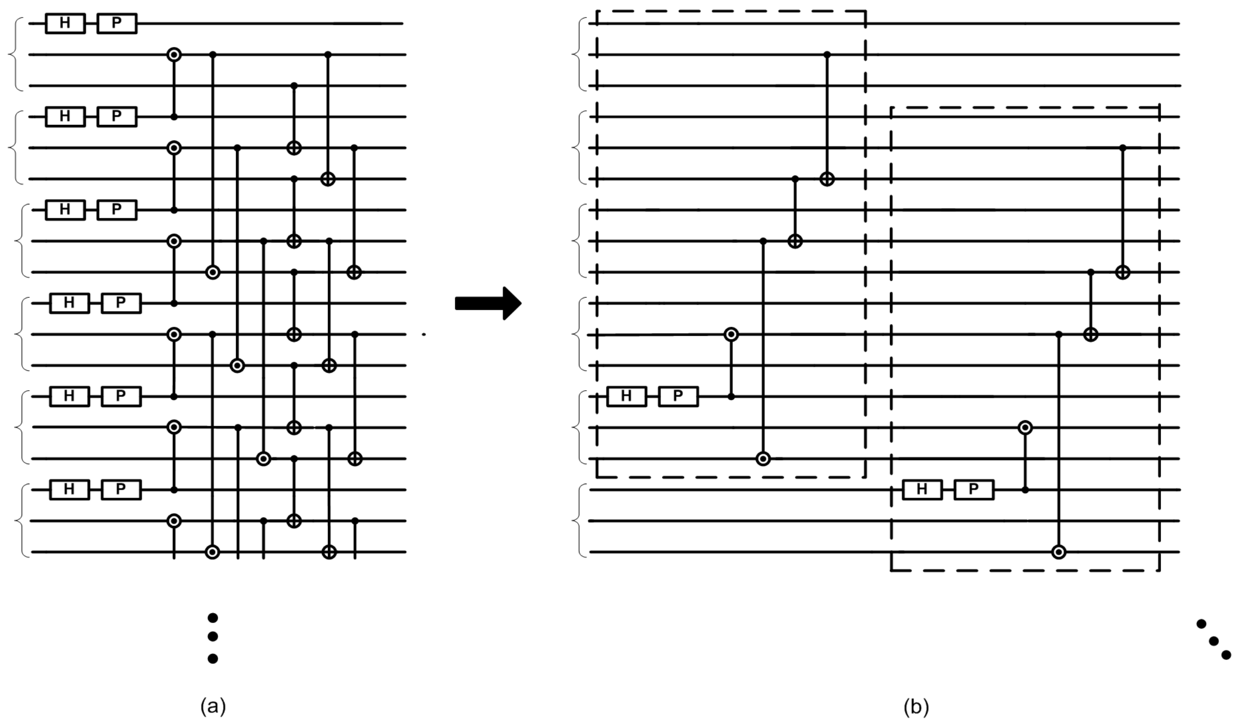

Figure 4: (a) pearl-necklace representation, and (b) minimal-memory convolutional encoder representation for example 1.

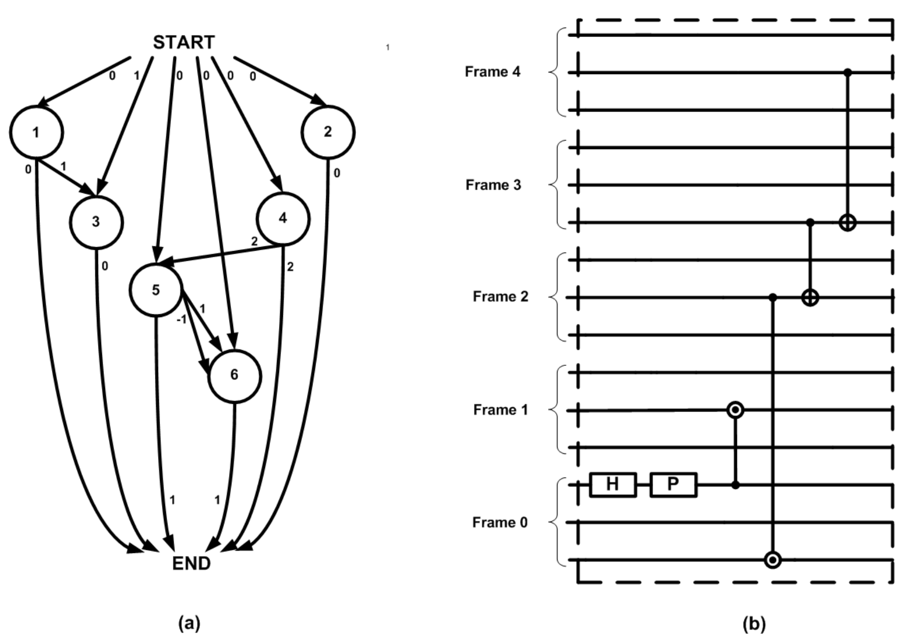

Figure 5: (a) The non-commutativity graph $\mathcal{G}$ and (b) a minimal-memory convolutional encoder for Example 1

with weight equal to four (0+2+1+1). Therefore the minimal memory for the convolutional encoder is equal to four frames of memory qubits. Also from inspecting the graph $\mathcal{G}$, we can determine the locations for all the target qubit frame indices: $\tau_1 = 0$, $\tau_2 = 0$, $\tau_3 = 1$, $\tau_4 = 0$, $\tau_5 = 2$, and $\tau_6 = 3$. Figure 5(b) depicts a minimal-memory convolutional encoder for this example. Figure 4(b) depicts minimal-memory convolutional encoder representation for the pearl-necklace encoder in Figure 4(a).

3 Conclusion

In this paper, we have proposed an algorithm to find a practical realization with a minimal memory requirement for a a pearl-necklace encoder of a general quantum convolutional code, which includes any gate string in the shift-invariant Clifford group. We have shown that the non-commutativity relations of gate strings in the encoder determine the realization. We introduce a non-commutativity graph, whose each vertex corresponds to a gate string in the pearl-necklace encoder. The weighted edges represent non-commutativity relations in the encoder. Using the graph, the minimal-memory realization can be obtained. The weight of the longest path in the graph is equal to the minimal required memory of the encoder. The running time for the construction of the graph and finding the longest path is quadratic in the number of gate strings in the pearl-necklace encoder.

As we mentioned in our previous paper [1], an open question still remains. The proposed algorithm begins with a particular pearl-necklace encoder, and finds the minimal required memory for it. But one can start with polynomial description of convolutional code and find the minimal required memory for the code. There are two problems here to work on: (1) finding the pearl-necklace encoder with minimal-memory requirements among all pearl-necklace encoders that implement the same code, (2) constructing a repeated unitary directly from the polynomial description of the code itself, and attempting to minimize the memory requirements of realizing this code.

Acknowledgements

The authors would like to thank Mark M. Wilde for his helpful discussions, comments and careful reading of an earlier version of this paper.

References

[1] Monireh Houshmand, Saied Hosseini-Khayat, and Mark M.Wilde. Minimal memory requirements for pearl necklace encoders of quantum convolutional codes. *arXiv:1004.5179*, 2010.

[2] Frank Gaitan. *Quantum Error Correction and Fault Tolerant Quantum Computing*. CRC Press, Taylor and Francis Group, 2008.

[3] Daniel A. Lidar, Isaac L. Chuang, and K. Birgitta Whaley. Decoherence-free subspaces for quantum computation. *Physical Review Letters*, 81(12):2594–2597, September 1998.

[4] Daniel Gottesman. *Stabilizer Codes and Quantum Error Correction*. PhD thesis, California Institute of Technology, May 1997. arXiv:quant-ph/9705052.

[5] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[6] Andrew M. Steane. Error correcting codes in quantum theory. *Physical Review Letters*, 77(5):793–797, July 1996.

[7] A. Robert Calderbank and Peter W. Shor. Good quantum error-correcting codes exist. *Physical Review A*, 54(2):1098–1105, August 1996.

[8] A. Robert Calderbank, Eric M. Rains, Peter W. Shor, and N. J. A. Sloane. Quantum error correction and orthogonal geometry. *Physical Review Letters*, 78(3):405–408, January 1997.

[9] A. Robert Calderbank, Eric M. Rains, Peter W. Shor, and N. J. A. Sloane. Quantum error correction via codes over GF(4). *IEEE Transactions on Information Theory*, 44:1369–1387, 1998.

[10] Paolo Zanardi and Mario Rasetti. Noiseless quantum codes. *Physical Review Letters*, 79(17):3306–3309, October 1997.

[11] Paolo Zanardi and Mario Rasetti. Error avoiding quantum codes. *Modern Physics Letters B*, 11:1085–1093, 1997.

[12] Harold Ollivier and Jean-Pierre Tillich. Description of a quantum convolutional code. *Physical Review Letters*, 91(17):177902, October 2003.

[13] Harold Ollivier and Jean-Pierre Tillich. Quantum convolutional codes: Fundamentals. *arXiv:quant-ph/0401134*, 2004.

[14] Markus Grassl and Martin Rötteler. Noncatastrophic encoders and encoder inverses for quantum convolutional codes. In *Proceedings of the IEEE International Symposium on Information Theory*, pages 1109–1113, Seattle, Washington, USA, July 2006. arXiv:quant-ph/0602129.

[15] Markus Grassl and Martin Rötteler. Quantum convolutional codes: Encoders and structural properties. In *Proceedings of the Forty-Fourth Annual Allerton Conference*, pages 510–519, Allerton House, UIUC, Illinois, USA, September 2006.

[16] Markus Grassl and Martin Rötteler. Constructions of quantum convolutional codes. In *Proceedings of the IEEE International Symposium on Information Theory*, pages 816–820, Nice, France, June 2007. arXiv:quant-ph/0703182.

[17] G. David Forney and Saikat Guha. Simple rate-1/3 convolutional and tail-biting quantum error-correcting codes. In *IEEE International Symposium on Information Theory (arXiv:quant-ph/0501099)*, pages 1028–1032, September 2005.

[18] G. David Forney, Markus Grassl, and Saikat Guha. Convolutional and tail-biting quantum error-correcting codes. *IEEE Transactions on Information Theory*, 53:865–880, 2007.

[19] Salah A. Aly, Markus Grassl, Andreas Klappenecker, Martin Rötteler, and Pradeep Kiran Sarvepalli. Quantum convolutional BCH codes. In *10th Canadian Workshop on Information Theory (arXiv:quant-ph/0703113)*, pages 180–183, 2007.

[20] Salah A. Aly, Andreas Klappenecker, and Pradeep Kiran Sarvepalli. Quantum convolutional codes derived from Reed-Solomon and Reed-Muller codes. In *International Symposium on Information Theory (arXiv:quant-ph/0701037)*, pages 821–825, Nice, France, June 2007.

[21] Mark M. Wilde, Hari Krovi, and Todd A. Brun. Convolutional entanglement distillation. In *Proceedings of the IEEE International Symposium on Information Theory*, Austin, Texas, USA, June 2010. arXiv:0708.3699.

[22] Mark M. Wilde and Todd A. Brun. Entanglement-assisted quantum convolutional coding. *To appear in Physical Review A*, 2010. arXiv:0712.2223.

[23] Mark M. Wilde and Todd A. Brun. Unified quantum convolutional coding. In *Proceedings of the IEEE International Symposium on Information Theory*, pages 359–363, Toronto, Ontario, Canada, July 2008. arXiv:0801.0821.

[24] Mark M. Wilde and Todd A. Brun. Quantum convolutional coding with shared entanglement: General structure. *To appear in Quantum Information Processing*, 2010. arXiv:0807.3803.

[25] Mark M. Wilde and Todd A. Brun. Extra shared entanglement reduces memory demand in quantum convolutional coding. *Physical Review A*, 79(3):032313, March 2009.

[26] David Poulin, Jean-Pierre Tillich, and Harold Ollivier. Quantum serial turbo-codes. *IEEE Transactions on Information Theory*, 55(6):2776–2798, June 2009. arXiv:0712.2888.

[27] Mark M. Wilde. Quantum-shift-register circuits. *Physical Review A*, 79(6):062325, June 2009.

[28] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. MIT Press, 2009.