2-1-2012

Minimal-memory requirements for pearl-necklace encoders of quantum convolutional codes

Monireh Houshmand
Ferdowsi University of Mashhad

Saied Hosseini-Khayat
Ferdowsi University of Mashhad

Mark M. Wilde
Université McGill

Recommended Citation

Houshmand, M., Hosseini-Khayat, S., & Wilde, M. (2012). Minimal-memory requirements for pearl-necklace encoders of quantum convolutional codes. *IEEE Transactions on Computers, 61* (3), 299-312. https://doi.org/10.1109/TC.2010.226

Minimal memory requirements for pearl-necklace encoders of quantum convolutional codes

Monireh Houshmand, Saied Hosseini-Khayat, and Mark M. Wilde

Abstract

One of the major goals in quantum information processing is to reduce the overhead associated with the practical implementation of quantum protocols, and often, routines for quantum error correction account for most of this overhead. A particular technique for quantum error correction that may be useful for protecting a stream of quantum information is quantum convolutional coding. The encoder for a quantum convolutional code has a representation as a convolutional encoder or as a "pearl-necklace" encoder. In the pearl-necklace representation, it has not been particularly clear in the research literature how much quantum memory such an encoder would require for implementation. Here, we offer an algorithm that answers this question. The algorithm first constructs a weighted, directed acyclic graph where each vertex of the graph corresponds to a gate string in the pearl-necklace encoder, and each path through the graph represents a path through non-commuting gates in the encoder. We show that the weight of the longest path through the graph is equal to the minimal amount of memory needed to implement the encoder. A dynamic programming search through this graph determines the longest path. The running time for the construction of the graph and search through it is quadratic in the number of gate strings in the pearl-necklace encoder.

Index Terms

quantum communication, quantum convolutional codes, quantum shift register circuits, quantum error correction, quantum memory

Quantum information science [1] is an interdisciplinary field combining quantum physics, mathematics and computer science. Quantum computers give dramatic speedups over classical ones for tasks such as integer factorization [2] and database search [3]. Two parties can also securely agree on a secret key by exploiting certain features of quantum mechanics [4].

A quantum system intracts with its environment, and this interaction inevitably alters the state of the quantum system, which causes loss of information encoded in it. Quantum error correction [1], [5], [6] offers a way to combat

- *M. Houshmand and S. Hosseini-Khayat are with the Department of Electrical Engineering, Ferdowsi University of Mashhad, Iran. E-mail: monirehhoushmand@gmail.com; saied.hosseini@gmail.com*
- *M. M. Wilde is a postdoctoral fellow with the School of Computer Science, McGill University, Montreal, Québec, Canada. E-mail: mark.wilde@mcgill.ca*

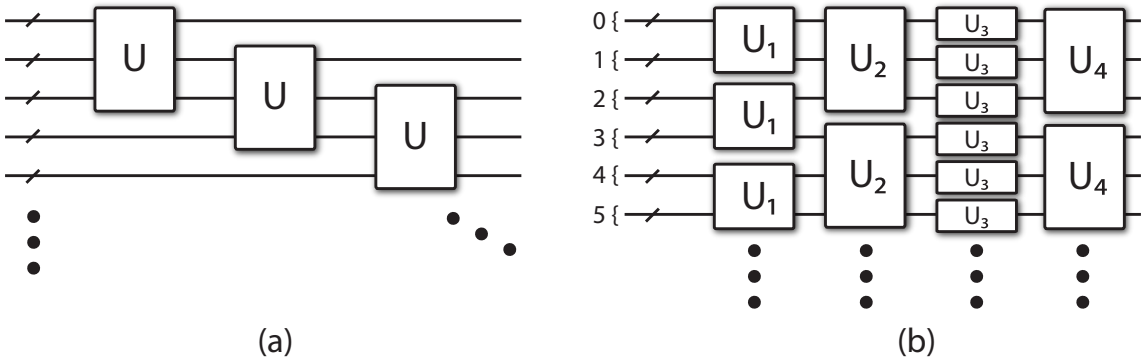

Fig. 1. Two different representations of the encoder for a quantum convolutional code. (a) Representation of the encoder as a convolutional encoder. (b) Representation of the encoder as a pearl-necklace encoder. The numbering at the inputs of the pearl-necklace encoder indicates our convention for frame indexing.

this noise—it is the fundamental theory underpinning the practical realization of quantum computation and quantum communication. The routines associated with it will account for most of the overhead in the implementation of several practical quantum protocols. Thus, any reduction in the overhead or resources for implementing quantum error correction should aid in building a practical quantum system. One example of such a resource is the size of a quantum memory needed to implement the routines of quantum error correction.

A quantum convolutional code is a particular quantum code that protects a stream of quantum information communicated over a quantum channel [7], [8]. These codes are inspired by their classical counterparts [9] and inherit many of their properties: they admit a mathematical description in terms of a parity check matrix of binary polynomials or binary rational functions and have a memory structure. They also have low-complexity encoding and decoding circuits and an efficient maximum likelihood error estimation procedure helps estimate errors under the assumption that they are transmitted over a memoryless channel [7], [10], [11].

One representation of the encoder for a quantum convolutional code has a simple form [7]. It consists of a single unitary repeatedly applied to a stream of quantum data—we call such a form for the encoder a *convolutional encoder* (see Figure 1(a)). An important practical concern for the implementation of an encoder is the amount of quantum storage or memory it requires. The representation of the encoder in the convolutional form allows one to determine this quantity in a straightforward manner: 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 encoder in Figure 3(c) requires three memory qubits for implementation. Ollivier and Tillich pursued this approach for encoding in their early work on quantum convolutional codes [7], [11], and more recently, Poulin, Ollivier, and Tillich exploited this approach in their construction of quantum turbo codes [10]. They randomly generated and filtered Clifford unitaries at random to act as the convolutional encoders for the constituent quantum convolutional codes of a quantum turbo code. In this case, it was straightforward to determine the memory required to implement a quantum turbo code because they represented the two constituent encoders for the quantum turbo code in the convolutional form.

An alternate representation for the encoder of a quantum convolutional code consists of several "strings" of the

same unitary applied to the quantum data stream (see Figure 1(b)). This representation of the encoder is known as a *pearl-necklace encoder* due to its striking similarity to a pearl necklace (each string of unitaries corresponds to one bead of a pearl necklace in this analogy). Ollivier and Tillich coined this term and realized the importance of this structure early on [7], [11], while Grassl and Rötteler (GR) later constructed detailed encoding algorithms for encoding quantum convolutional codes with pearl-necklace encoders [12]. 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. Grassl and Rötteler then showed how to produce a quantum convolutional code from two dual-containing classical binary convolutional codes (much like the Calderbank-Shor-Steane or CSS approach [13], [14]) and then constructed a pearl-necklace encoder for the produced code [15]. Later work demonstrated how to produce an entanglement-assisted quantum convolutional code from two *arbitrary* classical binary convolutional codes and, in some cases, it is possible to construct a pearl-necklace encoder for the produced quantum code [16], [17], [18], [19], [20]. The advantage of the Grassl-Rötteler (GR) and the subsequent entanglement-assisted approach is that the quantum code designer can choose high-performance classical convolutional codes to import for use as high-performance quantum convolutional codes.

The representation of a GR pearl-necklace encoder as a convolutional encoder was originally somewhat unclear, but a recent paper demonstrated how to translate between these two representations[1] [21]. There, the author exploited notions from linear system theory to show how convolutional encoders realize the transformations in the GR pearl-necklace encoders.[2] An important contribution of Ref. [21] was to clarify the notion of quantum memory in a GR pearl-necklace encoder, given that such a notion is not explicitly clear in the pearl-necklace representation. In fact, Ref. [21] demonstrates that a particular convolutional encoder for the Forney-Grassl-Guha code [8] requires fives frame of memory qubits, whereas an earlier analysis of Grassl and Rötteler suggested that this encoder would require only two frames [12].

The goal of the present paper is to outline an algorithm that computes the memory requirements of a GR pearl-necklace encoder.[3] Our approach considers a class of potential convolutional encoders that realize the same transformation that a GR pearl-necklace encoder does. The ideas are in the same spirit as those in Ref. [21], but the approach here is different. The algorithm to compute the memory requirements then consists of two parts:

1) It first constructs a weighted, directed acyclic graph where each vertex of the graph corresponds to a string of gates in the GR pearl-necklace encoder. The graph features a directed edge from one vertex to another if the two corresponding gate strings do not commute, and the weight of a directed edge depends on the degrees of the two corresponding gate strings. Thus, the graph details paths through non-commuting gates in the pearl-necklace encoder. The complexity for constructing this graph is quadratic in the number of gate strings

1. The author of Ref. [21] called a convolutional encoder a "quantum shift register circuit" to make contact with the language of classical shift registers, but the two terms are essentially interchangeable.

2. Perhaps Ref. [21] is the most explicit work to show why quantum convolutional codes are in fact "convolutional."

3. Ref. [21] suggested a formula as an upper bound on the memory requirements for the Grassl-Rötteler pearl-necklace encoder of a Calderbank-Shor-Steane (CSS) code, but subsequent analysis demonstrates this upper bound does not hold for all encoders. The algorithm in the present paper is able to determine the exact memory requirements of a given Grassl-Rötteler pearl-necklace encoder.

in the GR pearl-necklace encoder.

2) We show that the longest path of the graph corresponds to the minimal amount of memory that a convolutional encoder requires, and the final part of the algorithm finds this longest path through the graph with dynamic programming [22]. This final part has complexity linear in the number of vertices and edges in the graph (or, equivalently, quadratic in the number of gate strings in the pearl-necklace encoder) because the graph is directed and acyclic.

In this paper, we focus on encoding CSS quantum convolutional codes, for which each elementary operation corresponds to a string of CNOT gates in a pearl-necklace encoder (see Section VI of Ref. [21]). A later work addresses the general case of non-CSS quantum convolutional codes [23]. We begin with a particular pearl-necklace encoder of a quantum convolutional code and determine the minimal amount of quantum memory needed to implement it as a convolutional encoder.

We structure this work as follows. First we review some basic concepts from quantum mechanics. Section 2 then establishes some definitions and notation that we employ throughout this paper. Our main contribution begins in Section 3. We first determine the memory requirements for some simple examples of pearl-necklace encoders. We then build up to more complicated examples, by determining the memory required for convolutional encoders with CNOT gates that are unidirectional, unidirectional in the opposite direction, and finally with arbitrary direction. The direction is with respect to the source and target qubits of the CNOT gates in the convolutional encoder (for example, the convolutional encoder in Figure 6(b) is unidirectional). The final section of the paper concludes with a summary and suggestions for future research.

1 QUANTUM STATES AND GATES

The basic data unit in a quantum computer is the qubit. A *qubit* is a unit vector in a two dimensional Hilbert space, $\mathcal{H}_2$ for which a particular basis, denoted by $|0\rangle$, $|1\rangle$, has been fixed. The basis states $|0\rangle$ and $|1\rangle$ are quantum analogues of classical 0 and 1 respectively. Unlike classical bits, qubits can be in a superposition of $|0\rangle$ and $|1\rangle$ such as $a\,|0\rangle + b\,|1\rangle$ where a and b are complex numbers such that $|a|^2 + |b|^2 = 1$. If such a superposition is measured with respect to the basis $|0\rangle$,$|1\rangle$, then $|0\rangle$ is observed with probability $|a|^2$ and $|1\rangle$ is observed with probability $|b|^2$.

An n-qubit register is a quantum system whose state space is $\mathcal{H}_2^n$. Given the computational basis $\{|0\rangle, |1\rangle\}$ for $\mathcal{H}_2$, the basis states of this register are in the following set:

$$\{|i_1\rangle \otimes |i_2\rangle \otimes \cdots \otimes |i_n\rangle\,; i_1, i_2, \cdots, i_n = 0, 1\},$$

or equivalently

$$\{|i_1 i_2 \cdots i_n\rangle\,; i_1, i_2, \cdots, i_n = 0, 1\}.$$

The state $|\psi\rangle$ of an n-qubit register is a vector in 2^n-dimensional space:

$$|\psi\rangle = \sum_{i_1,i_2,\cdots,i_n=0,1} a_{i_1,i_2,\cdots,i_n} |i_1\rangle \otimes |i_2\rangle \otimes \cdots \otimes |i_n\rangle ,$$

where

$$\sum_{i_1,i_2,\cdots,i_n=0,1} |a_{i_1,i_2,\cdots,i_n}|^2 = 1.$$

The phenomenon of *quantum entanglement* [1], which has no classical analogue, has been recognized as an important physical resource in many areas of quantum computation and quantum information science. A multi-qubit quantum state $|\psi\rangle$ is said to be entangled if it cannot be written as the tensor product $|\psi\rangle = |\phi_1\rangle \otimes |\phi_2\rangle$ of two pure states. For example, the EPR pair shown below is an entangled quantum state:

$$|\Phi\rangle = (|00\rangle + |11\rangle)/\sqrt{2}.$$

In other words, in the case of an entangled state, the qubits are linked in a way such that one cannot describe the quantum state of a constituent of the system independent of its counterparts, even if the individual qubits are spatially separated.

As with classical circuits, quantum operations can be performed by networks of gates. Every quantum gate is a linear transformation represented by a unitary matrix, defined on an n-qubit Hilbert space. A matrix U is *unitary* if $UU^\dagger = I$, where $U^\dagger$ is the conjugate transpose of the matrix U. Since any unitary operation has an inverse, any quantum gate is reversible, meaning that given the state of a set of output qubits, it is possible to determine the state of its corresponding set of input qubits.

Some examples of useful single-qubit gates are the elements of the Pauli set Π. The set $\Pi = \{I, X, Y, Z\}$ consists of the Pauli operators:

$$I \equiv \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \; X \equiv \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \; Y \equiv \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \; Z \equiv \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}.$$

I is the identity transformation, X is a bit flip (NOT), Z is a phase flip operation, and Y is a combination of both. Two other important single-qubit transformations are the Hadamard gate H and phase gate P where

$$H \equiv \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix},$$

$$P \equiv \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}.$$

The n-qubit Pauli group Π^n is defined as n-fold tensor products of Pauli operators:

$$\Pi^n = \left\{ e^{i\phi} A_1 \otimes \cdots \otimes A_n : \forall j \in \{1, \ldots, n\}, \quad A_j \in \Pi, \quad \phi \in \{0, \pi/2, \pi, 3\pi/2\} \right\}. \tag{1}$$

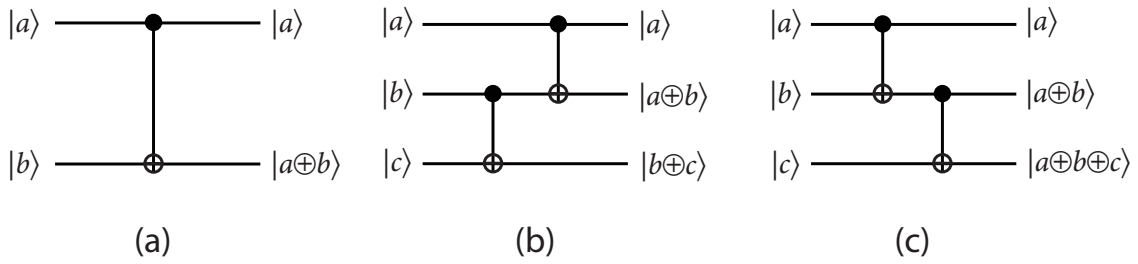

Fig. 2. (a) the circuit representation of CNOT gate. (b) shows that when index of the source qubit of one gate is the same as the index of the target of the other, two CNOTs do not commute.

The controlled-NOT gate (CNOT) gate is a two-qubit gate. The first qubit serves as a control and the second as a target. CNOT performs the NOT operation on the target qubit if the control qubit is $|1\rangle$ and otherwise leaves it unchanged. In other words, the second output is the XOR of the target and control qubit. The matrix representation of the CNOT gate is

$$\text{CNOT} \equiv \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

Figure 2(a) shows the circuit representation of the CNOT gate. The non-commutativity of CNOT gates is the most important concept needed to understand this paper. Two CNOT gates do not commute if the index of the source qubit of one is the same as the index of the target of the other. Figure 2 (b) and (c) show this fact. If the input to both circuits is $|a\rangle \otimes |b\rangle \otimes |c\rangle$ $(a, b, c = 0, 1)$, the output of the circuit depicted in Figure 2 (b) is $|a\rangle \otimes |a \oplus b\rangle \otimes |b \oplus c\rangle$, while the output of the circuit depicted in Figure 2 (c) is $|a\rangle \otimes |a \oplus b\rangle \otimes |a \oplus b \oplus c\rangle$. Therefore the third qubits of two circuits are different when $a = 1$.

Thus, there are two kinds of non-commutativity in which we are interested for two gate strings in a pearl-necklace encoder:

1) *Source-target non-commutativity* occurs when 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. This type of non-commutativity occurs in the following two gate strings:

$$\overline{\text{CNOT}}(i, j)(D^{l_1})\ \overline{\text{CNOT}}(k, i)(D^{l_2}),$$

where we order the gate strings from left to right.

2) *Target-source non-commutativity* occurs when 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. For example, this type of non-commutativity occurs in the following two gate strings:

$$\overline{\text{CNOT}}(i, j)(D^{l_1})\ \overline{\text{CNOT}}(j, k)(D^{l_2}).$$

More generally, if U is a gate that operates on a single qubit where

$$U \equiv \begin{bmatrix} u_{00} & u_{01} \\ u_{10} & u_{11} \end{bmatrix},$$

then the controlled-U gate is a gate that operates on two qubits in such a way that the first qubit serves as a control and the second as a target. This gate performs the unitary U on the target qubit if the control qubit is $|1\rangle$ and otherwise leaves it unchanged. The matrix representation of the controlled-U gate is:

$$\text{Controlled-U} \equiv \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & u_{00} & u_{01} \\ 0 & 0 & u_{10} & u_{11} \end{bmatrix}.$$

A special type of unitary matrix which is often used in the encoding circuit of a quantum error correction code is called a Clifford operation [1]. A Clifford operation U is one that preserves elements of the Pauli group under conjugation: $A \in \Pi^n \Rightarrow UAU^\dagger \in \Pi^n$. The CNOT gate, the Hadamard gate H, and the phase gate P suffice to implement any unitary matrix in the Clifford group [6].

2 Definitions and Notation

We first establish some definitions and notation before proceeding with the main development of the paper.

Our convention for numbering the frames in a pearl-necklace encoder is from "top" to "bottom." In contrast, our convention for numbering the frames upon which the unitary of a convolutional encoder acts is from "bottom" to "top." Figure 1(b) illustrates the former convention for a pearl-necklace encoder, while Figure 6(b) illustrates the later convention for a convolutional encoder. These conventions are useful for our analysis later on.

We now establish conventions for indexing gates in pearl-necklace encoders. Let s_k and t_k denote the frame index of the respective source and target qubits of a gate in the k^{th} gate string of a pearl-necklace encoder. For example, consider the pearl-necklace encoder in Figure 3(a) that has two gate strings. The index $k = 1$ for the left gate string, and $k = 2$ for the right gate string. The second CNOT gate in the $k = 1$ gate string has $s_1 = 1$ and $t_1 = 1$. The third CNOT gate in the $k = 2$ gate string has $s_2 = 2$ and $t_2 = 3$.

We also require some conventions for indexing gates in a convolutional encoder. Let σ_k and τ_k denote the frame index of the respective source and target qubits of the k^{th} gate in a convolutional encoder. For example, consider the convolutional encoder in Figure 6(b). The third gate from the left has $k = 3$, $\sigma_3 = 2$, and $\tau_3 = 0$.

Whether referring to a pearl-necklace encoder or a convolutional encoder, the notation $\text{CNOT}(i,j)\,(s,t)$ denotes a CNOT gate from qubit i in frame s to qubit j in frame t. We employ this notation extensively in what follows. The notation $\overline{\text{CNOT}}\,(i,j)\left(D^l\right)$ refers to a string of gates in a pearl-necklace encoder and denotes an infinite, repeated sequence of CNOT gates from qubit i to qubit j in every frame where qubit j is in a frame delayed by l. For example,

the left string of gates in Figure 3(a) corresponds to $\overline{\text{CNOT}}\,(1,2)\,(1)$, while the right string of gates corresponds to $\overline{\text{CNOT}}\,(1,3)\,(D)$.

The following two Boolean functions are useful later on in our algorithms for computing memory requirements:

$$\text{Source-Target}\left(\overline{\text{CNOT}}(a_1,b_1)(D^{l_1}),\overline{\text{CNOT}}(a_2,b_2)(D^{l_2})\right),$$
$$\text{Target-Source}\left(\overline{\text{CNOT}}(a_1,b_1)(D^{l_1}),\overline{\text{CNOT}}(a_2,b_2)(D^{l_2})\right).$$

The first function takes two gate strings $\overline{\text{CNOT}}(a_1,b_1)(D^{l_1})$ and $\overline{\text{CNOT}}(a_2,b_2)(D^{l_2})$ as input. It returns TRUE if $\overline{\text{CNOT}}(a_1,b_1)(D^{l_1})$ and $\overline{\text{CNOT}}(a_2,b_2)(D^{l_2})$ have source-target non-commutativity (i.e., $a_1 = b_2$) and returns FALSE otherwise. The second function also takes two gate strings $\overline{\text{CNOT}}(a_1,b_1)(D^{l_1})$ and $\overline{\text{CNOT}}(a_2,b_2)(D^{l_2})$ as input. It returns TRUE if $\overline{\text{CNOT}}(a_1,b_1)$ and $\overline{\text{CNOT}}(a_2,b_2)$ have target-source non-commutativity (i.e., $a_1 = b_2$) and returns FALSE otherwise.

The following succession of N gate strings realizes a pearl-necklace encoder:

$$\overline{\text{CNOT}}\,(a_1,b_1)\left(D^{l_1}\right)\ \overline{\text{CNOT}}\,(a_2,b_2)\left(D^{l_2}\right)\ \cdots\ \overline{\text{CNOT}}\,(a_N,b_N)\left(D^{l_N}\right). \tag{2}$$

Consider the j^{th} gate string $\overline{\text{CNOT}}\,(a_j,b_j)\left(D^{l_j}\right)$ in the above succession of N gate strings. It is important to consider the gate strings preceding this one that have source-target non-commutativity with it, target-source non-commutativity with it, non-negative degree, and negative degree. This leads to four different subsets $\mathcal{S}_j^+$, $\mathcal{S}_j^-$, $\mathcal{T}_j^+$, and $\mathcal{T}_j^-$ that we define as follows:

$$\mathcal{S}_j^+ = \{i \mid \text{Source-Target}(\overline{\text{CNOT}}\,(a_i,b_i)\left(D^{l_i}\right),\overline{\text{CNOT}}\,(a_j,b_j)\left(D^{l_j}\right)) = \text{TRUE}, i \in \{1,2,\cdots,j-1\}, l_i \geq 0\},$$
$$\mathcal{S}_j^- = \{i \mid \text{Source-Target}(\overline{\text{CNOT}}\,(a_i,b_i)\left(D^{l_i}\right),\overline{\text{CNOT}}\,(a_j,b_j)\left(D^{l_j}\right)) = \text{TRUE}, i \in \{1,2,\cdots,j-1\}, l_i < 0\},$$
$$\mathcal{T}_j^+ = \{i \mid \text{Target-Source}(\overline{\text{CNOT}}\,(a_i,b_i)\left(D^{l_i}\right),\overline{\text{CNOT}}\,(a_j,b_j)\left(D^{l_j}\right)) = \text{TRUE}, i \in \{1,2,\cdots,j-1\}, l_i \geq 0\},$$
$$\mathcal{T}_j^- = \{i \mid \text{Target-Source}(\overline{\text{CNOT}}\,(a_i,b_i)\left(D^{l_i}\right),\overline{\text{CNOT}}\,(a_j,b_j)\left(D^{l_j}\right)) = \text{TRUE}, i \in \{1,2,\cdots,j-1\}, l_i < 0\}.$$

The first subset $\mathcal{S}_j^+$ consists of all the non-negative-degree gate strings preceding gate j that have source-target non-commutativity with it. The second subset $\mathcal{S}_j^-$ consists of all the negative-degree gate strings preceding gate j that have source-target non-commutativity with it. The third subset $\mathcal{T}_j^+$ consists of all the non-negative-degree gate strings preceding gate j that have target-source non-commutativity with it. The fourth subset $\mathcal{T}_j^-$ consists of all the negative-degree gate strings preceding gate j that have target-source non-commutativity with it. We use these subsets extensively in what follows.

3 Memory Requirements for Pearl-Necklace Encoders

The first step in determining the memory requirements for a GR pearl-necklace encoder is to rearrange it as a convolutional encoder. There are many possible correct candidates for the convolutional encoder ("correct" in the sense that they encode the same code), but there is a realization that uses a minimal amount of memory qubits. This

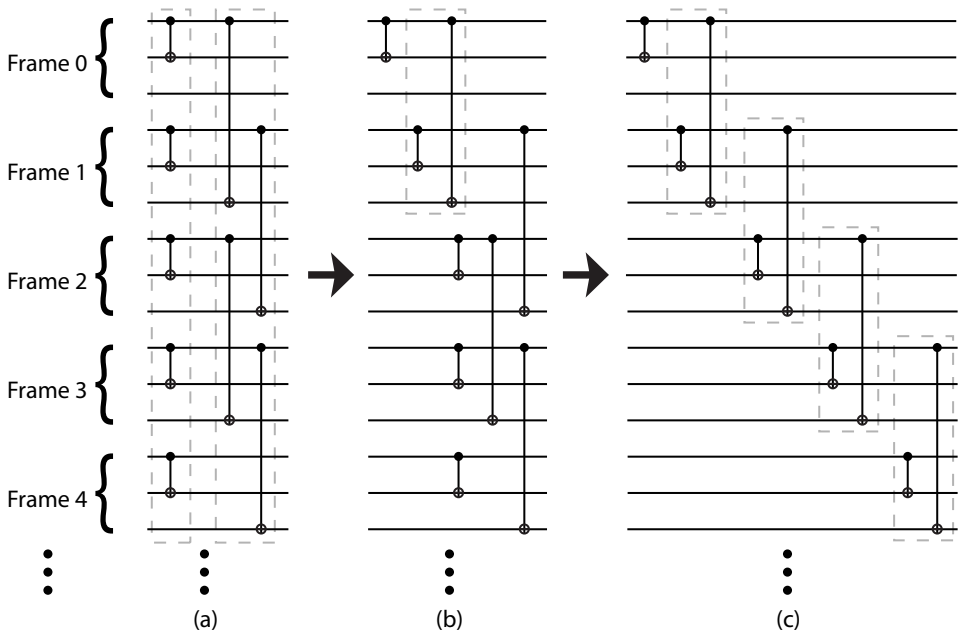

Fig. 3. Simple depiction of the rearrangement of a pearl-necklace encoder into a convolutional encoder (note that the technique is more complicated than depicted for more general cases). (a) The pearl-necklace encoder consists of the gate strings $\overline{\text{CNOT}}\,(1,2)\,(1)\;\overline{\text{CNOT}}\,(1,3)\,(D)$. (b) The rearrangement of the first few gates by shifting the gates below the first three to the right. (c) A convolutional encoder realization of the pearl-necklace encoder in (a). The repeated application of the procedure in (b) realizes a convolutional encoder from a pearl-necklace encoder.

idea of rearrangement is in the same spirit as some of the original ideas of Ollivier and Tillich where there they were trying to determine non-catastrophic encoders [7], [11], but here we explicitly apply them to the GR pearl-necklace encoder for the purpose of determining memory requirements. In order to make a convolutional encoder, 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 of its remaining gates commute with this set. Then we can shift all the gates remaining in the pearl-necklace encoder to the right and infinitely repeat this operation on the remaining gates. Figure 3 shows a simple example of the rearrangement of a pearl-necklace encoder $\overline{\text{CNOT}}(1,2)\,(1)\,\overline{\text{CNOT}}(1,3)(D)$ into a convolutional encoder.

3.1 The source-target constraint and the target-source constraint

We begin by explaining some constraints that apply to convolutional encoders formed from primitive pearl-necklace encoders. First consider a pearl-necklace encoder that is a succession of m CNOT gate strings:

$$\overline{\text{CNOT}}\,(a_1,b_1)\left(D^{l_1}\right)\;\overline{\text{CNOT}}\,(a_2,b_2)\left(D^{l_2}\right)\;\cdots\;\overline{\text{CNOT}}\,(a_m,b_m)\left(D^{l_m}\right).$$

Suppose that all the gate strings in the above succession commute with each other, in the sense that $a_i \neq b_j$ for all $i \neq j$. Then the candidates for a convolutional encoder are members of the following set M:

$$M \equiv \{\text{CNOT}(a_1,b_1)(s_1,t_1)\;\cdots\;\text{CNOT}(a_m,b_m)(s_m,t_m) : t_i = s_i + l_i,\;\; i \in \{1,\ldots,m\},\;\; s_i \in \{0\} \cup \mathbb{N}\}\,, \tag{3}$$

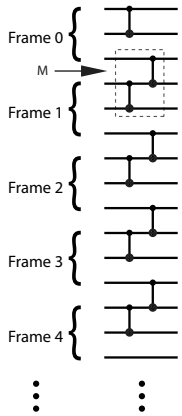

Fig. 4. A correct sample M for the encoder $\overline{\text{CNOT}}\,(1,2)\left(D^0\right)\overline{\text{CNOT}}\,(3,1)\left(D^1\right)$.

where $\mathbb{N}=\{1,2,\ldots\}$. All members of M are correct choices for a convolutional encoder because they produce the required encoding and because all the remaining gates in the pearl-necklace encoder commute with a particular element of M in all cases. Thus, there is no constraint on each frame index s_i of the source qubit of the i^{th} CNOT gate.

Now suppose that two CNOT gates in the pearl-necklace encoder do not commute with each other. Recall that this non-commutativity occurs in two ways:

1) Source-target non-commutativity occurs in the following two gate strings:

$$\overline{\text{CNOT}}\,(i,j)\left(D^{l_1}\right)\ \overline{\text{CNOT}}\,(k,i)\left(D^{l_2}\right), \tag{4}$$

where $j\neq k$. Potential candidates for a convolutional encoder belong to the following set M:

$$M\equiv\{\text{CNOT}(i,j)(s_1,t_1)\ \text{CNOT}(k,i)(s_2,t_2):t_1=s_1+l_1,\ t_2=s_2+l_2,\ s_1,s_2\in\{0\}\cup\mathbb{N}\},$$

though some choices in the set M may not be correct because they ignore the non-commutativity of the gate strings in (4). In order for the convolutional encoder to be correct, we should choose the frame indices s_1 and t_2 such that all the gates in the gate string $\overline{\text{CNOT}}\,(i,j)\left(D^{l_1}\right)$ that remain after $\text{CNOT}(i,j)(s_1,t_1)$ commute with the gate $\text{CNOT}(k,i)(s_2,t_2)$. Otherwise, the chosen convolutional encoder implements the transformation in (4) in the opposite order. An an example, Figure 4 shows the gate strings $\overline{\text{CNOT}}\,(1,2)\left(D^0\right)\overline{\text{CNOT}}\,(3,1)\left(D^1\right)$. A correct sample candidate M for the encoder is

$$M\equiv\text{CNOT}(1,2)(1,1)\text{CNOT}(3,1)(0,1),$$

which is shown in the figure. It is obvious that the gates remaining after $\text{CNOT}\,(1,2)\,(1,1)$ (the highlighted gates) commute with $\text{CNOT}(3,1)(0,1)$.

The gate $\text{CNOT}(i,j)(t_2,t_2+l_1)$ is the only gate in the gate string $\overline{\text{CNOT}}\,(i,j)\left(D^{l_1}\right)$ that does not commute with $\text{CNOT}(k,i)(s_2,t_2)$. Thus, this gate cannot belong to the remaining set of gates. The set of all gates in the gate string

$\overline{\text{CNOT}}\,(i,j)\left(D^{l_1}\right)$ remaining after a particular gate CNOT$(i,j)(s_1,t_1)$ is as follows:

$$\{\text{CNOT}(i,j)(s_1+d,t_1+d) : d \in \mathbb{N}\}. \tag{5}$$

The following inequality determines a restriction on the source qubit frame index s_1 such that the gates in the above set both commute with CNOT$(k,i)(s_2,t_2)$ and lead to a correct convolutional encoder:

$$\forall d \in \mathbb{N} \quad s_1 + d > t_2, \tag{6}$$

because these are the remaining gates that we can shift to the right. Furthermore, the following inequality applies to any correct choice of the first gate in a convolutional encoder because this gate can be either CNOT$(i,j)(t_2,t_2+l_1)$ or any other in the set in (5) that obeys the inequality in (6):

$$s_1 \geq t_2. \tag{7}$$

The inequality in (7) is the *source-target constraint* and applies to any correct choice of a convolutional encoder that implements the transformation in (4).

2) The second case is similar to the above case with a few notable changes. Target-source non-commutativity occurs in the following two gate strings:

$$\overline{\text{CNOT}}\,(i,j)\left(D^{l_1}\right)\ \overline{\text{CNOT}}\,(j,k)\left(D^{l_2}\right). \tag{8}$$

Potential candidates for a convolutional encoder belong to the following set M where

$$M \equiv \{\text{CNOT}(i,j)(s_1,t_1)\ \text{CNOT}(j,k)(s_2,t_2) : t_1 = s_1 + l_1,\ t_2 = s_2 + l_2,\ s_1, s_2 \in \{0\} \cup \mathbb{N}\},$$

though some choices in the set M may not be correct because they ignore the non-commutativity of the gate strings in (8). In order for the convolutional encoder to be correct, we should choose the frame indices t_1 and s_2 such that the gates in the gate string $\overline{\text{CNOT}}\,(i,j)\left(D^{l_1}\right)$ that remain after CNOT$(i,j)(s_1,t_1)$ commute with CNOT$(j,k)(s_2,t_2)$. Otherwise, the chosen convolutional encoder implements the transformation in (8) in the opposite order. The gate CNOT$(i,j)(s_2-l_1,s_2)$ is the only gate in $\overline{\text{CNOT}}\,(i,j)\left(D^{l_1}\right)$ that does not commute with CNOT$(j,k)(s_2,t_2)$. Thus, this gate cannot belong to the remaining set of gates. The set of all gates in the gate string $\overline{\text{CNOT}}\,(i,j)\left(D^{l_1}\right)$ remaining after a particular gate CNOT$(i,j)(s_1,t_1)$ is as follows:

$$\{\text{CNOT}(i,j)(s_1+d,t_1+d) : d \in \mathbb{N}\}. \tag{9}$$

The following inequality determines a restriction on the target qubit frame index t_1 such that the gates in the above set both commute with CNOT$(j,k)(s_2,t_2)$ and lead to a correct convolutional encoder:

$$\forall d \in \mathbb{N} \quad t_1 + d > s_2, \tag{10}$$

because these are the remaining gates that we can shift to the right. Furthermore, the following inequality applies to any correct choice of the first gate in a convolutional encoder because this gate can be either CNOT$(i, j)(s_2 - l_1, s_2)$ or any other in the set in (9) that obeys the inequality in (10):

$$t_1 \geq s_2. \tag{11}$$

The inequality in (11) is the *target-source constraint* and applies to any correct choice of a convolutional encoder that implements the transformation in (8).

3.2 Memory requirements for a unidirectional pearl-necklace encoder

We are now in a position to introduce our algorithms for finding a minimal-memory convolutional encoder that realizes the same transformation as a pearl-necklace encoder. In this subsection, we consider the memory requirements for a CSS pearl-necklace encoder with unidirectional CNOT gates (see Figure 6(b) for an example). Section 3.3 determines them for a CSS pearl-necklace encoder with unidirectional CNOT gates in the opposite direction, and Section 3.4 determines them for a general CSS pearl-necklace encoder with CNOT gates in an arbitrary direction.

First consider a pearl-necklace encoder that is a sequence of several CNOT gate strings:

$$\overline{\text{CNOT}}\,(a_1, b_1)\left(D^{l_1}\right)\ \overline{\text{CNOT}}\,(a_2, b_2)\left(D^{l_2}\right)\ \cdots\ \overline{\text{CNOT}}\,(a_m, b_m)\left(D^{l_m}\right),$$

where all $l_i \geq 0$ and all the gate strings in the above succession commute with each other. All members of M in (3) are correct choices for the convolutional encoder, as explained in the beginning of Section 3.1. Though, choosing the same value for each target qubit frame index t_i results in the minimal required memory L where

$$L = \max\{l_1, l_2, ..., l_m\}.$$

A correct, minimal-memory choice for a convolutional encoder is as follows:

$$\text{CNOT}(a_1, b_1)(l_1, 0)\ \text{CNOT}(a_2, b_2)(l_2, 0)\ \cdots\ \text{CNOT}(a_m, b_m)(l_m, 0),$$

where we recall the convention that frames in the convolutional encoder number from "bottom" to "top."

Now consider two gate strings in a pearl-necklace encoder that have source-target non-commutativity:

$$\overline{\text{CNOT}}\,(i, j)\left(D^{l_1}\right)\ \overline{\text{CNOT}}\,(k, i)\left(D^{l_2}\right), \tag{12}$$

where $l_1, l_2 \geq 0$. Thus, the source-target constraint in (7) holds for any correct choice of a convolutional encoder. Choosing $s_1 = t_2$ leads to a minimal-memory convolutional encoder because any other choice either does not implement the correct transformation (it violates the source-target constraint) or it uses more memory than this

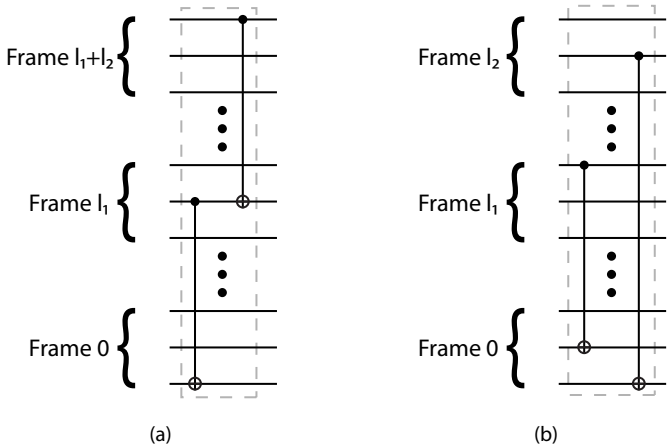

Fig. 5. Depiction of (a) a minimal-memory convolutional encoder for the gate strings $\overline{\text{CNOT}}\,(2,3)\left(D^{l_1}\right)\ \overline{\text{CNOT}}\,(1,2)\left(D^{l_2}\right)$, and (b) a minimal-memory convolutional encoder for the gate strings $\overline{\text{CNOT}}\,(1,2)\left(D^{l_1}\right)\ \overline{\text{CNOT}}\,(2,3)\left(D^{l_2}\right)$ where l_1 and l_2 are non-negative.

choice. So a correct, minimal-memory choice for a convolutional encoder is as follows:

$$\text{CNOT}\,(i,j)\,(l_1,0)\ \ \text{CNOT}\,(k,i)\,(l_1+l_2,l_1)\,.$$

Such a convolutional encoder requires L frames of memory qubits where

$$L = l_1 + l_2.$$

Figure 5(a) depicts a minimal-memory convolutional encoder for the following gate strings

$$\overline{\text{CNOT}}\,(2,3)\left(D^{l_1}\right)\ \overline{\text{CNOT}}\,(1,2)\left(D^{l_2}\right),$$

where l_1 and l_2 are both non-negative.

Consider two gate strings in a pearl-necklace encoder that have target-source non-commutativity:

$$\overline{\text{CNOT}}\,(i,j)\left(D^{l_1}\right)\ \overline{\text{CNOT}}\,(j,k)\left(D^{l_2}\right), \tag{13}$$

where $l_1, l_2 \geq 0$. Thus, the target-source constraint in (11) holds for any correct choice of a convolutional encoder. Choosing $t_1 = t_2$ leads to a minimal-memory convolutional encoder because any other choice either does not implement the correct transformation (it violates the target-source constraint) or it uses more memory than this choice. A correct, minimal-memory choice for the convolutional encoder is as follows:

$$\text{CNOT}(i,j)(l_1,0)\ \text{CNOT}(j,k)(l_2,0),$$

and the number L of frames of memory qubits it requires is as follows:

$$L = \max\{l_1, l_2\}.$$

Figure 5(b) depicts a minimal-memory convolutional encoder for the following gate strings

$$\overline{\text{CNOT}}\,(1,2)\left(D^{l_1}\right)\ \overline{\text{CNOT}}\,(2,3)\left(D^{l_2}\right),$$

where both l_1 and l_2 are non-negative.

Suppose that two gate strings have both kinds of non-commutativity:

$$\overline{\text{CNOT}}(i,j)(D^{l_1})\ \overline{\text{CNOT}}(j,i)(D^{l_2}),$$

where $l_1, l_2 \geq 0$. Thus, both constraints in (7) and (11) hold for any correct choice of a convolutional encoder. The target-source constraint in (11) holds if the source-target constraint in (7) holds. So it is sufficient to consider only the source-target constraint in such a scenario.

The above examples prepare us for constructing a minimal-memory convolutional encoder that implements the same transformation as a pearl-necklace encoder with unidirectional CNOT gates. Suppose that a pearl-necklace encoder features the following succession of N gate strings:

$$\overline{\text{CNOT}}\,(a_1,b_1)\left(D^{l_1}\right)\ \overline{\text{CNOT}}\,(a_2,b_2)\left(D^{l_2}\right)\ \cdots\ \overline{\text{CNOT}}\,(a_N,b_N)\left(D^{l_N}\right), \tag{14}$$

where all $l_i \geq 0$. The first gate in the convolutional encoder is $\text{CNOT}(a_1,b_1)\,(\sigma_1 = l_1, \tau_1 = 0)$. For the target indices of each gate j where $2 \leq j \leq N$, we should find the minimal value of τ_j that satisfies all the source-target and target-source constraints that the gates preceding it impose. The inequality in (15) applies to the target index of the j^{th} gate in the convolutional encoder by applying the source-target constraint in (7):

$$\begin{aligned} \sigma_i \leq \tau_j \quad & \forall i \in \mathcal{S}_j^+, \\ \therefore\ \tau_i + l_i \leq \tau_j \quad & \forall i \in \mathcal{S}_j^+, \\ \therefore\ \max\{\tau_i + l_i\}_{i \in \mathcal{S}_j^+} \leq \tau_j, & \end{aligned} \tag{15}$$

Recall that the direction of frame numbering in the convolutional encoder is opposite to the direction of numbering in the pearl-necklace encoder—so the direction of inequalities are reversed with respect to (7) and (11). The inequality in (15) exploits all of the source-target constraints corresponding to the gates preceding gate j in order to place a limit on the location of the j^{th} gate in the convolutional encoder. The inequality in (16) similarly exploits all of the target-source constraints corresponding to the gates preceding gate j:

$$\begin{aligned} \tau_i \leq \sigma_j \quad & \forall i \in \mathcal{T}_j^+, \\ \therefore\ \tau_i - l_j \leq \tau_j \quad & \forall i \in \mathcal{T}_j^+, \\ \therefore\ \max\{\tau_i - l_j\}_{i \in \mathcal{T}_j^+} \leq \tau_j. & \end{aligned} \tag{16}$$

The following constraint applies to the frame index τ_j of the target qubit of the j^{th} gate in the convolutional encoder,

by applying (15) and (16):

$$\tau_j \geq \max\{\{\tau_i + l_i\}_{i\in\mathcal{S}_j^+}, \{\tau_i - l_j\}_{i\in\mathcal{T}_j^+}\}.$$

Thus, the minimal value for τ_j that satisfies all the constraints is

$$\tau_j = \max\{\{\tau_i + l_i\}_{i\in\mathcal{S}_j^+}, \{\tau_i - l_j\}_{i\in\mathcal{T}_i^+}\}. \tag{17}$$

Of course, there is no constraint for the frame index τ_j if the gate string $\overline{\text{CNOT}}(a_j, b_j)(D^{l_j})$ commutes with all previous gate strings. Thus, in this case, we choose the frame index τ_j as follows:

$$\tau_j = 0. \tag{18}$$

A good choice for the frame index τ_j is as follows:

$$\tau_j = \max\{0, \{\tau_i + l_i\}_{i\in\mathcal{S}_j^+}, \{\tau_i - l_j\}_{i\in\mathcal{T}_j^-}\}, \tag{19}$$

by considering (17) and (18).

3.2.1 Construction of the commutativity graph

We introduce the notion of a *commutativity* graph in order to find the values in (19) 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 (14). Let G^+ denote the commutativity graph for a succession of gate strings that have purely non-negative degrees (and thus where the CNOT gates are unidirectional). Algorithm 1 below presents pseudo code for constructing the commutativity graph G^+.

Algorithm 1 Algorithm for determining the commutativity graph G^+ for purely non-negative 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 CNOT-bar(a_j, b_j)(D^{l_j})
  DrawEdge(START, j, 0)
  for i := 1 to j − 1 do
    if Source-Target(CNOT-bar(a_i, b_i)(D^{l_i}), CNOT-bar(a_j, b_j)(D^{l_j})) = TRUE then
      DrawEdge(i, j, l_i)
    else if Target-Source(CNOT-bar(a_i, b_i)(D^{l_i}), CNOT-bar(a_j, b_j)(D^{l_j})) = TRUE then
      DrawEdge(i, j, −l_j)
    end if
  end for
end for
Draw an END vertex
for j := 1 to N do
  DrawEdge(j, END, l_j)
end for
```

The commutativity graph G^+ consists of N vertices, labeled $1, 2, \cdots, N$, where the j^{th} vertex corresponds to the j^{th} gate string $\overline{\text{CNOT}}(a_j, b_j)(D^{l_j})$. 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. A zero-weight edge connects the START vertex to every vertex, and an l_j-weight edge connects every vertex j to the END vertex. Also, an l_i-weight edge connects the i^{th} vertex to the j^{th} vertex if

$$\text{Source-Target}\left(\overline{\text{CNOT}}(a_i, b_i)(D^{l_i}), \overline{\text{CNOT}}(a_j, b_j)(D^{l_j})\right) = \text{TRUE},$$

and a $-l_j$-weight edge connects the i^{th} vertex to the j^{th} vertex if

$$\text{Target-Source}\left(\overline{\text{CNOT}}(a_i, b_i)(D^{l_i}), \overline{\text{CNOT}}(a_j, b_j)(D^{l_j})\right) = \text{TRUE}.$$

The commutativity graph 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 construction of G^+ requires time quadratic in the number of gate strings in the pearl-necklace encoder. In Algorithm 1, the **if** instruction 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

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

3.2.2 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 commutativity graph G^+ is equal to 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 commutativity graph G^+ is

$$w_j = \tau_j. \tag{20}$$

A zero-weight edge connects the START vertex to the first vertex, so that $w_1 = \tau_1 = 0$. Thus the base step holds for the target index of the first CNOT gate in a minimal-memory convolutional encoder. Now suppose the property holds for the target indices of the first k CNOT gates in the convolutional encoder:

$$w_j = \tau_j \quad \forall j : 1 \leq j \leq k. \tag{21}$$

Suppose we add a new gate string $\overline{\text{CNOT}}(a_{k+1}, b_{k+1})(D^{l_{k+1}})$ to the pearl-necklace encoder, and Algorithm 1 then adds a new vertex $k+1$ to the graph G^+ and the following edges to G^+:

1) A zero-weight edge from the START vertex to vertex $k+1$.
2) An l_{k+1}-weight edge from vertex $k+1$ to the END vertex.
3) An l_i-weight edge from each vertex $\{i\}_{i \in \mathcal{S}_{k+1}^+}$ to vertex $k+1$.

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

So it is clear that the following relations hold because w_{k+1} is the weight of the longest path to vertex $k+1$ and from applying (21):

$$\begin{aligned} w_{k+1} &= \max\{0, \{w_i + l_i\}_{i\in\mathcal{S}^+_{k+1}}, \{w_i - l_{k+1}\}_{i\in\mathcal{T}^+_{k+1}}\}, \\ &= \max\{0, \{\tau_i + l_i\}_{i\in\mathcal{S}^+_{k+1}}, \{\tau_i - l_{k+1}\}_{i\in\mathcal{T}^+_{k+1}}\}. \end{aligned} \tag{22}$$

The inductive proof then follows by applying (19) and (22):

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

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

$$\begin{aligned} w &= \max_{i\in\{1,2,\cdots,N\}} \{w_i + l_i\} \\ &= \max_{i\in\{1,2,\cdots,N\}} \{\tau_i + l_i\} \\ &= \max_{i\in\{1,2,\cdots,N\}} \{\sigma_i\}. \end{aligned}$$

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 from the i^{th} vertex to the END vertex. The second equality follows by applying (20). The final equality follows because $\sigma_i = \tau_i + l_i$. The quantity $\max\{\sigma_i\}_{i\in\{1,2,\cdots,N\}}$ is equal to minimal required memory for a minimal-memory convolutional encoder because the largest location of a source qubit determines the number of frames upon which a convolutional encoder with unidirectional CNOT gates acts. (Recall that we number the frames starting from zero). Thus, the theorem holds. □

The final task is to determine the longest path in G^+. Finding the longest path in a general graph is an NP-complete problem, but dynamic programming finds it on a weighted, directed acyclic graph in time linear in the number of vertices and edges, or equivalently, quadratic in the number of gate strings in the pearl-necklace encoder [22].

3.2.3 Example of a pearl-necklace encoder with unidirectional CNOT gates

We conclude this development with an example.

Example 1. Consider the following gate strings in a pearl-necklace encoder:

$$\overline{\text{CNOT}}(2,3)(D)\ \overline{\text{CNOT}}(1,2)(D)\ \overline{\text{CNOT}}(2,3)(D^2)\ \overline{\text{CNOT}}(1,2)(1)\ \overline{\text{CNOT}}(2,1)(D).$$

All gate strings in the above pearl-necklace encoder have non-negative degree and are thus unidirectional. Figure 6(a) draws G^+ for this pearl-necklace encoder, after running Algorithm 1. The graph displays all of the source-target and target-source non-commutativities between gate strings in the pearl-necklace encoder. The longest path through

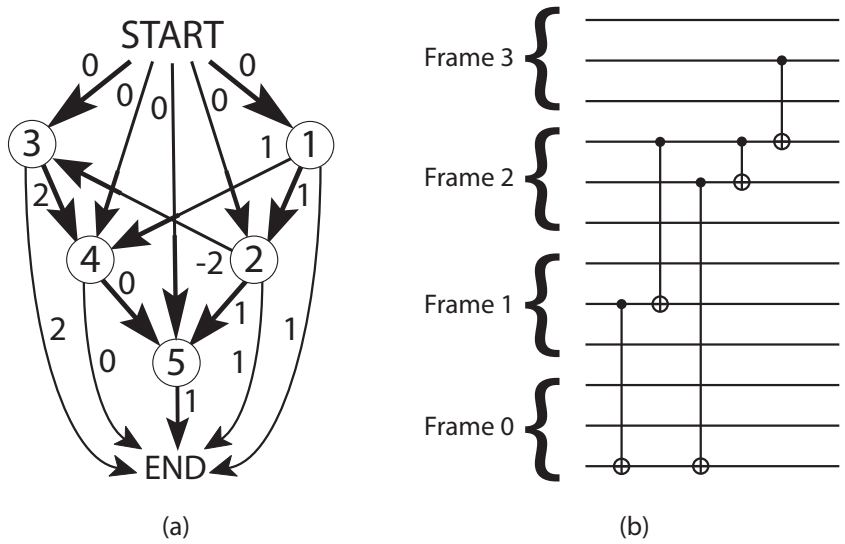

Fig. 6. (a) The commutativity graph G^+ and (b) a minimal-memory convolutional encoder for Example 1.

the graph is

$$\text{START} \to 3 \to 4 \to 5 \to \text{END},$$

with weight equal to three. So the minimal memory for the convolutional encoder is equal to three frames of memory qubits. Also from inspecting the graph G^+, we can determine the locations for all the target qubit frame indices: $\tau_1 = 0$, $\tau_2 = 1$, $\tau_3 = 0$, $\tau_4 = 2$, and $\tau_5 = 2$. Figure 6(b) depicts a minimal-memory convolutional encoder that implements the same transformation as the pearl-necklace encoder.

3.3 Memory requirements for a unidirectional pearl-necklace encoder in the opposite direction

In this section, we find a minimal-memory convolutional encoder that implements the same transformation as a pearl-necklace encoder with purely non-positive degree CNOT gates. The ideas in this section are similar to those in the previous one.

First consider a pearl-necklace encoder that is a succession of several CNOT gate strings:

$$\overline{\text{CNOT}}\left(a_1, b_1\right)\left(D^{l_1}\right)\ \overline{\text{CNOT}}\left(a_2, b_2\right)\left(D^{l_2}\right)\ \cdots\ \overline{\text{CNOT}}\left(a_m, b_m\right)\left(D^{l_m}\right),$$

where all $l_i \leq 0$ and all the gate strings commute with each other. All members of M in (3) are correct choices for the convolutional encoder, as explained in the beginning of Section 3.1. But this time, choosing the same value for each source qubit frame index s_i results in the minimal required memory L where

$$L = \max\{|l_1|, |l_2|, ..., |l_m|\}.$$

A correct choice for a minimal-memory convolutional encoder is

$$\text{CNOT}(a_1, b_1)(0, |l_1|)\ \text{CNOT}(a_2, b_2)(0, |l_2|)\ \cdots\ \text{CNOT}(a_m, b_m)(0, |l_m|).$$

Now consider two gate strings that have source-target non-commutativity:

$$\overline{\text{CNOT}}\,(i,j)\left(D^{l_1}\right)\ \overline{\text{CNOT}}\,(k,i)\left(D^{l_2}\right), \tag{23}$$

where $l_1, l_2 \leq 0$. Thus, the source-target constraint in (7) holds for any correct choice of a convolutional encoder. Choosing $s_1 = s_2$ leads to the minimal memory required for the convolutional encoder because any other choice either does not implement the correct transformation (it violates the source-target constraint) or it uses more memory than this choice. A correct choice for a minimal-memory convolutional encoder is

$$\text{CNOT}\,(i,j)\,(0,|l_1|)\ \text{CNOT}\,(k,i)\,(0,|l_2|)\,.$$

Such a convolutional encoder requires L frames of memory qubits where

$$L = \max\{|l_1|\,,|l_2|\}.$$

Figure 7(a) illustrates a minimal-memory convolutional encoder for the gate strings

$$\overline{\text{CNOT}}\,(3,2)\left(D^{l_1}\right)\ \overline{\text{CNOT}}\,(1,3)\left(D^{l_2}\right),$$

where $l_1, l_2 \leq 0$.

Now consider two gate strings that have target-source non-commutativity:

$$\overline{\text{CNOT}}\,(i,j)\left(D^{l_1}\right)\overline{\text{CNOT}}\,(j,k)\left(D^{l_2}\right), \tag{24}$$

that where $l_1, l_2 \leq 0$. The target-source constraint in (11) holds for any correct choice of a convolutional encoder. Choosing $t_1 = s_2$ leads to a minimal-memory convolutional encoder because any other choice either does not implement the correct transformation (it violates the target-source constraint) or it requires more memory than this choice. A correct choice for a minimal-memory convolutional encoder is

$$\text{CNOT}\,(i,j)\,(0,|l_1|)\,\text{CNOT}\,(k,i)\,(|l_1|\,,|l_1+l_2|)\,,$$

with the number L of frames of memory qubits as follows:

$$L = |l_1 + l_2|\,.$$

Figure 7(b) shows a minimal-memory convolutional encoder for the encoding sequence

$$\overline{\text{CNOT}}\,(3,2)\left(D^{l_1}\right)\ \overline{\text{CNOT}}\,(2,1)\left(D^{l_2}\right),$$

where l_1 and l_2 are non-positive.

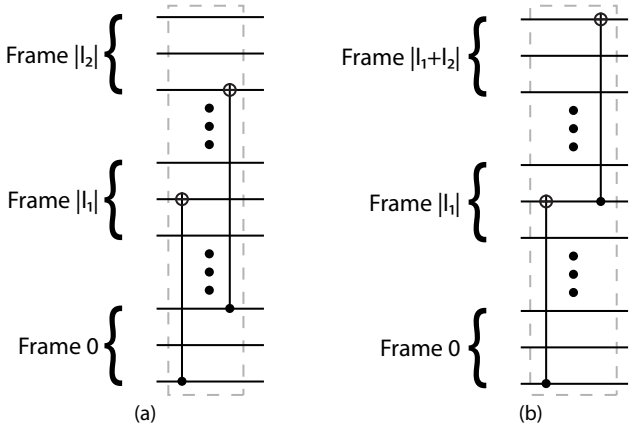

Fig. 7. (a) A minimal-memory convolutional encoder for the gate strings $\overline{\text{CNOT}}\,(3,2)\left(D^{l_1}\right)\ \overline{\text{CNOT}}\,(1,3)\left(D^{l_2}\right)$, and (b) a minimal-memory convolutional encoder for the gate strings $\overline{\text{CNOT}}\,(3,2)\left(D^{l_1}\right)\ \overline{\text{CNOT}}\,(2,1)\left(D^{l_2}\right)$ where l_1 and l_2 are non-positive.

Suppose we have two gate strings that feature both types of non-commutativity:

$$\overline{\text{CNOT}}(i,j)(D^{l_1})\ \overline{\text{CNOT}}(j,i)(D^{l_2}).$$

Thus, both constraints in (7) and (11) hold for any correct choice of a convolutional encoder. The source-target constraint in (7) holds if the target-source constraint in (11) holds when both degrees are non-positive. So it is sufficient to consider only the target-source constraint in this scenario.

The above examples prepare us for constructing a minimal-memory convolutional encoder that implements the same transformation as a pearl-necklace encoder with unidirectional CNOT gates (the gates are in the opposite direction of those in Section 3.2). Suppose that a pearl-necklace encoder features the following succession of N gate strings:

$$\overline{\text{CNOT}}\,(a_1,b_1)\left(D^{l_1}\right)\ \overline{\text{CNOT}}\,(a_2,b_2)\left(D^{l_2}\right)\ \cdots\ \overline{\text{CNOT}}\,(a_N,b_N)\left(D^{l_N}\right), \tag{25}$$

where N is the number of gate strings and all $l_i \leq 0$. The first gate in the convolutional encoder is $\text{CNOT}(\sigma_1 = 0, \tau_1 = l_1)$. For the source indices of gate j where $2 \leq j \leq N$, we should find the minimal value for σ_j that satisfies all the source-target and target-source constraints that the previous gates impose. The following inequalities apply to the source qubit frame index σ_j of the j^{th} gate in the convolutional encoder:

$$\begin{aligned} \sigma_i &\leq \tau_j \quad \forall i \in \mathcal{S}_j^-, \\ \therefore\ \sigma_i &\leq \sigma_j + |l_j| \quad \forall i \in \mathcal{S}_j^-, \\ \therefore\ \max\{\sigma_i - |l_j|\}_{i \in \mathcal{S}_j^-} &\leq \sigma_j, \end{aligned} \tag{26}$$

The inequality in (26) exploits all of the source-target constraints corresponding to the gates preceding gate j in order to place a limit on the location of the j^{th} gate in the convolutional encoder. The inequality below similarly

exploits all of the target-source constraints corresponding to the gates preceding gate j:

$$\begin{aligned} \tau_i \leq \sigma_j \quad \forall i \in \mathcal{T}_j^-, \\ \therefore \ \ \sigma_i + |l_i| \leq \sigma_j \quad \forall i \in \mathcal{T}_j^-, \\ \therefore \ \ \max\{\sigma_i + |l_i|\}_{i \in \mathcal{T}_j^-} \leq \sigma_j. \end{aligned} \tag{27}$$

The following constraint applies to the frame index σ_j of the source qubit of the j^{th} gate in the convolutional encoder, by applying (26) and (27):

$$\sigma_j \geq \max\{\{\sigma_i - |l_j|\}_{i \in \mathcal{S}_j^-}, \{\sigma_i + |l_i|\}_{i \in \mathcal{T}_i^-}\}.$$

Thus, the minimal value for σ_j that satisfies all the constraints is

$$\sigma_j = \max\{\{\sigma_i - |l_j|\}_{i \in \mathcal{S}_j^-}, \{\sigma_i + |l_i|\}_{i \in \mathcal{T}_j^-}\} \tag{28}$$

There is no constraint for the source index σ_j if the gate string $\overline{\text{CNOT}}(a_j, b_j)(D^{l_j})$ commutes with all previous gate strings. Thus, in this case, we can choose σ_j as follows:

$$\sigma_j = 0. \tag{29}$$

So, based on (28) and (29), a good choice for σ_j is as follows:

$$\sigma_j = \max\{0, \{\sigma_i - |l_j|\}_{i \in \mathcal{S}_j^-}, \{\sigma_i + |l_i|\}_{i \in \mathcal{T}_j^-}\}. \tag{30}$$

3.3.1 Construction of the commutativity graph for the non-positive degree case

We construct a *commutativity* graph G^- in order to find the values in (30). It is again a weighted, directed acyclic graph constructed from the non-commutativity relations in the pearl-necklace encoder in (25). Algorithm 2 presents pseudo code for the construction of the commutativity graph in the non-positive degree case.

The graph G^- consists of N vertices, labeled $1, 2, \cdots, N$, where vertex j corresponds to the j^{th} gate string $\overline{\text{CNOT}}(a_j, b_j)(D^{l_j})$. A zero-weight edge connects the START vertex to all vertices, and an $|l_j|$-weight edge connects every vertex j to the END vertex. Also, an $|l_i|$-weight edge connects vertex i to vertex j if

$$\text{Target-Source}\left(\overline{\text{CNOT}}(a_i, b_i)(D^{l_i}), \overline{\text{CNOT}}(a_j, b_j)(D^{l_j})\right) = \text{TRUE},$$

and a $-|l_j|$-weight edge connects vertex i to vertex j if

$$\text{Source-Target}\left(\overline{\text{CNOT}}(a_i, b_i)(D^{l_i}), \overline{\text{CNOT}}(a_j, b_j)(D^{l_j})\right) = \text{TRUE}.$$

The graph G^- is an acyclic graph and its construction complexity is $O(N^2)$ (similar to the complexity for constructing G^+). Dynamic programming can find the longest path in G^- in time linear in the number of vertices and edges,

Algorithm 2 Algorithm for determining the commutativity graph G^- for purely non-positive case

$N \leftarrow$ 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 $\overline{\text{CNOT}}(a_j, b_j)(D^{l_j})$
  DrawEdge(**START**, j, 0)
  for $i := 1$ to $j-1$ **do**
    if Target-Source$\big(\overline{\text{CNOT}}(a_i, b_i)(D^{l_i}), \overline{\text{CNOT}}(a_j, b_j)(D^{l_j})\big) =$ **TRUE then**
      DrawEdge($i, j, |l_i|$)
    else if Source-Target$\big(\overline{\text{CNOT}}(a_i, b_i)(D^{l_i}), \overline{\text{CNOT}}(a_j, b_j)(D^{l_j})\big) =$ **TRUE then**
      DrawEdge($i, j, -|l_j|$)
    end if
  end for
end for
Draw an **END** vertex.
for $j := 1$ to N **do**
  DrawEdge(j,**END**,$|l_j|$)
end for

or equivalently, quadratic in the number of gate strings in the pearl-necklace encoder.

3.3.2 The longest path gives the minimal memory requirements

We now prove that the weight of the longest path from the START vertex to END vertex in G^- is equal to the memory in a minimal-memory realization of the pearl-necklace encoder in (25).

Theorem 2. *The weight of the longest path from the START vertex to END vertex in the graph G^- is equal to the minimal memory requirements of the convolutional encoder.*

Proof: By similar reasoning as in Theorem 1, the weight of the longest path from the START vertex to vertex j in the commutativity graph G^- is equal to

$$w_j = \sigma_j \tag{31}$$

Similar to the proof of Theorem 1, we can prove that the longest path from the START vertex to END vertex in G^- is equal to $\max\{\tau_i\}_{1 \le i \le N}$. Thus, it is equal to the minimal required number of frames of memory qubits. □

3.3.3 Example of a pearl-necklace encoder with unidirectional CNOT gates in the opposite direction

The following example illustrates how to find the minimal required memory for a purely non-positive degree pearl-necklace encoder.

Example 2. Consider the following succession of gate strings in a pearl-necklace encoder:

$$\overline{\text{CNOT}}(2,3)(D^{-1})\ \overline{\text{CNOT}}(1,2)(D^{-1})\ \overline{\text{CNOT}}(2,3)(D^{-2})\ \overline{\text{CNOT}}(1,2)(1)\ \overline{\text{CNOT}}(2,1)(D^{-1}).$$

All gates have non-positive powers and thus are unidirectional. Figure 8(a) illustrates the commutativity graph for this pearl-necklace encoder. The commutativity graph details all of the source-target and target-source non-

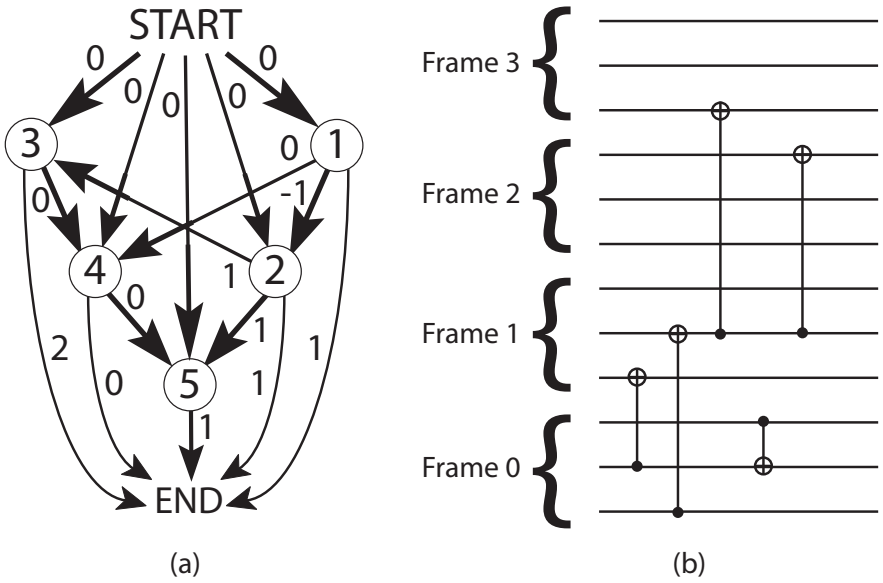

Fig. 8. (a) The commutativity graph G^+ and (b) a minimal-memory convolutional encoder for Example 2.

commutativities between gate strings. The longest path in G^- is

$$\text{START} \to 2 \to 3 \to \text{END},$$

with its weight equal to three. The memory needed to implement the convolutional encoder is three frames of memory qubits. From inspecting the commutativity graph G^-, we can also determine the locations of the source qubit frame indices: $\sigma_1 = 0$, $\sigma_2 = 0$, $\sigma_3 = 1$, $\sigma_4 = 0$, and $\sigma_5 = 1$. Figure 8(b) depicts a minimal-memory convolutional encoder for this example.

3.4 Memory requirements for an arbitrary CNOT pearl-necklace encoder

This section is the culmination of the previous two developments in Sections 3.2 and 3.3. Here, we find a minimal-memory convolutional encoder that implements the same transformation as a general pearl-necklace encoder with arbitrary CNOT gate strings.

Consider a pearl-necklace encoder that is a succession of several arbitrary CNOT gate strings:

$$\overline{\text{CNOT}}\left(a_1, b_1\right)\left(D^{l_1}\right)\ \overline{\text{CNOT}}\left(a_2, b_2\right)\left(D^{l_2}\right)\ \cdots\ \overline{\text{CNOT}}\left(a_N, b_N\right)\left(D^{l_N}\right). \tag{32}$$

We construct a commutativity graph G in order to determine a minimal-memory convolutional encoder. This graph is similar to those in Sections 3.2 and 3.3, but it combines ideas from both developments. In this graph, the weight of the longest path from the START vertex to vertex j is equal to τ_j when $l_j \geq 0$, and it is equal to σ_j when $l_j \leq 0$. We consider the constraints that the gates preceding gate j impose. The constraint inequalities use the target qubit frame index τ_j when $l_j \geq 0$ and use the source qubit frame index σ_j when $l_j < 0$. First consider the case when $l_j \geq 0$. The source-target and target-source constraints that previous gates impose on gate j occur in four

different ways, based on the sign of the involved gate's degree:

1) There is a source-target constraint for all gates preceding gate j that have non-negative degree and source-target non-commutativity with it:

$$\begin{aligned} \sigma_i \leq \tau_j \quad \forall i \in \mathcal{S}_j^+ \\ \therefore \ \ \tau_i + l_i \leq \tau_j \quad \forall i \in \mathcal{S}_j^+. \end{aligned}$$

2) There is a source-target constraint for all gates preceding gate j that have negative degree and source-target non-commutativity with it:

$$\sigma_i \leq \tau_j \quad \forall i \in \mathcal{S}_j^-.$$

3) There is a target-source constraint for all gates preceding gate j that have non-negative degree and target-source non-commutativity with it:

$$\begin{aligned} \tau_i \leq \sigma_j \quad \forall i \in \mathcal{T}_j^+ \\ \therefore \ \ \tau_i \leq \tau_j + l_j \quad \forall i \in \mathcal{T}_j^+, \\ \therefore \ \ \tau_i - l_j \leq \tau_j \quad \forall i \in \mathcal{T}_j^+. \end{aligned}$$

4) There is a target-source constraint for all gates preceding gate j that have negative degree and target-source non-commutativity with it:

$$\begin{aligned} \tau_i \leq \sigma_j \quad \forall i \in \mathcal{T}_j^- \\ \therefore \ \ \sigma_i + |l_i| \leq \tau_j + l_j \quad \forall i \in \mathcal{T}_j^-, \\ \therefore \ \ \sigma_i + |l_i| - l_j \leq \tau_j \quad \forall i \in \mathcal{T}_j^-. \end{aligned}$$

The graph includes an edge from vertex i to vertex j, corresponding to each of the above constraints. The target qubit frame index τ_j should satisfy the following inequality, by considering the above four inequalities:

$$\max\{\{\tau_i + l_i\}_{i \in \mathcal{S}_j^+}, \{\sigma_i\}_{i \in \mathcal{S}_j^-}, \{\tau_i - l_j\}_{i \in \mathcal{T}_j^+}, \{\sigma_i + |l_i| - l_j\}_{i \in \mathcal{T}_j^-}\} \leq \tau_j. \tag{33}$$

Choosing τ_j so that it minimally satisfies the above constraints results in a minimal usage of memory:

$$\tau_j = \max\{\{\tau_i + l_i\}_{i \in \mathcal{S}_j^+}, \{\sigma_i\}_{i \in \mathcal{S}_j^-}, \{\tau_i - l_j\}_{i \in \mathcal{T}_j^+}, \{\sigma_i + |l_i| - l_j\}_{i \in \mathcal{T}_j^-}\}. \tag{34}$$

There is no constraint for a gate string $\overline{\mathrm{CNOT}}(a_j, b_j)(D^{l_j})$ that commutes with its previous gates:

$$\tau_j = 0. \tag{35}$$

Thus choosing τ_j as follows when $l_j \geq 0$ results in minimal memory usage, based on (34) and (35):

$$\tau_j = \max\{0, \{\tau_i + l_i\}_{i\in\mathcal{S}_j^+}, \{\sigma_i\}_{i\in\mathcal{S}_j^-}, \{\tau_i - l_j\}_{i\in\mathcal{T}_j^+}, \{\sigma_i + |l_i| - l_j\}_{i\in\mathcal{T}_j^-}\}. \tag{36}$$

Now we consider the constraints that gates preceding gate j impose on it when $l_j < 0$. There are four different non-commutativity constraints based on the sign of the involved gate's degree:

1) There is a source-target constraint for all gates preceding gate j that have non-negative degree and source-target non-commutativity with it:

$$\begin{aligned} \sigma_i &\leq \tau_j \quad \forall i \in \mathcal{S}_j^+ \\ \therefore\ \tau_i + l_i &\leq \sigma_j + |l_j| \quad \forall i \in \mathcal{S}_j^+, \\ \therefore\ \tau_i + l_i - |l_j| &\leq \sigma_j \quad \forall i \in \mathcal{S}_j^+. \end{aligned}$$

2) There is a source-target constraint for all gates preceding gate j that have negative degree and source-target non-commutativity with it:

$$\begin{aligned} \sigma_i &\leq \tau_j \quad \forall i \in \mathcal{S}_j^- \\ \therefore\ \sigma_i &\leq \sigma_j + |l_j| \quad \forall i \in \mathcal{S}_j^- \\ \therefore\ \sigma_i - |l_j| &\leq \sigma_j \quad \forall i \in \mathcal{S}_j^-. \end{aligned}$$

3) There is a target-source constraint for all gates preceding gate j that have non-negative degree and target-source non-commutativity with it:

$$\tau_i \leq \sigma_j \quad \forall i \in \mathcal{T}_j^+.$$

4) There is a target-source constraint for all gates preceding gate j that have negative degree and target-source non-commutativity with it:

$$\begin{aligned} \tau_i &\leq \sigma_j \quad \forall i \in \mathcal{T}_j^+, \\ \therefore\ \sigma_i + |l_i| &\leq \sigma_j \quad \forall i \in \mathcal{T}_j^+. \end{aligned}$$

For similar reasons as above, choosing σ_j as follows results in minimal memory usage when $l_i < 0$:

$$\sigma_j = \max\{0, \{\tau_i + l_i - |l_j|\}_{i\in\mathcal{S}_j^+}, \{\sigma_i - |l_j|\}_{i\in\mathcal{S}_j^-}, \{\tau_i\}_{i\in\mathcal{T}_j^+}, \{\sigma_i + |l_i|\}_{i\in\mathcal{T}_j^-}\}. \tag{37}$$

A search through the constructed commutativity graph G can find the values in (36) and (37). Algorithm 3 below gives the pseudo code for constructing the commutativity graph G. The graph G consists of N vertices, labeled $1, 2, \cdots, N$, and vertex j corresponds to j^{th} gate string $\overline{\text{CNOT}}(a_j, b_j)(D^{l_j})$ in the pearl-necklace encoder.

Algorithm 3 Algorithm for determining the commutativity graph G in mixed case

$N \leftarrow$ 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} encoding operation, $\overline{\text{CNOT}}(a_j, b_j)(D^{l_j})$
  DrawEdge(**START**, j, 0)
  for $i := 1$ to $j-1$ **do**
    if $l_j \geq 0$ AND $l_i \geq 0$ **then**
      if Source-Target$\big(\overline{\text{CNOT}}(a_i, b_i)(D^{l_i}), \overline{\text{CNOT}}(a_j, b_j)(D^{l_j})\big) =$ **TRUE then**
        DrawEdge(i, j, l_i)
      else if Target-Source$\big(\overline{\text{CNOT}}(a_i, b_i)(D^{l_i}), \overline{\text{CNOT}}(a_j, b_j)(D^{l_j})\big) =$ **TRUE then**
        DrawEdge($i, j, -l_j$)
      end if
    else if $l_j \geq 0$ AND $l_i < 0$ **then**
      if Source-Target$\big(\overline{\text{CNOT}}(a_i, b_i)(D^{l_i}), \overline{\text{CNOT}}(a_j, b_j)(D^{l_j})\big) =$ **TRUE then**
        DrawEdge($i, j, 0$)
      end if
      if Target-Source$\big(\overline{\text{CNOT}}(a_i, b_i)(D^{l_i}), \overline{\text{CNOT}}(a_j, b_j)(D^{l_j})\big) =$ **TRUE then**
        DrawEdge($i, j, |l_i| - l_j$)
      end if
    else if $l_j < 0$ AND $l_i \geq 0$ **then**
      if Source-Target$\big(\overline{\text{CNOT}}(a_i, b_i)(D^{l_i}), \overline{\text{CNOT}}(a_j, b_j)(D^{l_j})\big) =$ **TRUE then**
        DrawEdge($i, j, l_i - |l_j|$)
      end if
      if Target-Source$\big(\overline{\text{CNOT}}(a_i, b_i)(D^{l_i}), \overline{\text{CNOT}}(a_j, b_j)(D^{l_j})\big) =$ **TRUE then**
        DrawEdge($i, j, 0$)
      end if
    else if $l_j < 0$ AND $l_i < 0$ **then**
      if Target-Source$\big(\overline{\text{CNOT}}(a_i, b_i)(D^{l_i}), \overline{\text{CNOT}}(a_j, b_j)(D^{l_j})\big) =$ **TRUE then**
        DrawEdge($i, j, |l_i|$)
      else if Source-Target$\big(\overline{\text{CNOT}}(a_i, b_i)(D^{l_i}), \overline{\text{CNOT}}(a_j, b_j)(D^{l_j})\big) =$ **TRUE then**
        DrawEdge($i, j, -|l_j|$)
      end if
    end if
  end for
end for
Draw an **END** vertex.
for $j := 1$ to N **do**
  DrawEdge(j, **END**, $|l_j|$)
end for

3.4.1 The longest path gives the minimal memory requirements

Theorem 3 states that the weight of the longest path from the START vertex to END vertex in G is equal to the minimal required memory for the encoding sequence in (32). Part of its proof follows from Lemma 4.

Theorem 3. *The weight of the longest path from the START vertex to the END vertex in the commutativity graph G is equal to the memory required for the convolutional encoder.*

Proof: The longest path in the graph is the maximum of the longest path to each vertex summed with the weight of the edge from each vertex to the end vertex:

$$w = \max\{w_j + |l_j|\}_{j \in \{1,2,\cdots,N\}},$$

The following relation holds by applying Lemma 4 below:

$$w = \max\{\{\tau_j + l_j\}_{l_j \geq 0}, \{\sigma_j + |l_j|\}_{l_j < 0}\},$$

which is equal to

$$w = \max\{\{\sigma_j\}_{l_j \geq 0}, \{\tau_j\}_{l_j < 0}\}.$$

The quantity $\max\{\{\sigma_j\}_{l_j \geq 0}, \{\tau_j\}_{l_j < 0}\}$ is equal to the memory requirement of a minimal-memory convolutional encoder, so the theorem holds. □

Lemma 4. The weight of the longest path from the START vertex to vertex j in G, w_j is equal to

$$w_j = \begin{cases} \tau_j & : \quad l_j \geq 0 \\ \sigma_j & : \quad l_j < 0 \end{cases}.$$

Proof: We prove the lemma by induction. The weight w_1 of the path to the first vertex is equal to zero because a zero-weight edge connects the START vertex to the first vertex. If $l_1 \geq 0$, then $\tau_1 = 0$. So $w_1 = \tau_1 = 0$ and if $l_1 < 0$, then $\sigma_1 = 0$. So $w_1 = \sigma_1 = 0$. Therefore the lemma holds for the first gate.

Suppose the lemma holds for the first k gates:

$$\forall i \in \{1, \cdots, k\}, \qquad w_i = \begin{cases} \tau_i & : \quad l_i \geq 0 \\ \sigma_i & : \quad l_i < 0 \end{cases}.$$

Consider adding a $(k+1)^{\text{th}}$ gate string $\overline{\text{CNOT}}(a_{k+1}, b_{k+1})(D^{l_{k+1}})$ to the pearl-necklace encoder. Algorithm 3 then adds a vertex with label $k+1$ to the graph. First consider the case that $l_{k+1} \geq 0$. Algorithm 3 then adds the following edges to the graph G:

1) A zero-weight edge from the START vertex to vertex $k+1$.
2) An l_{k+1}-weight edge from vertex $k+1$ to the END vertex.
3) An l_i-weight edge from each vertex $\{i\}_{i \in \mathcal{S}^+_{k+1}}$ to vertex $k+1$.
4) A zero-weight edge from each vertex $\{i\}_{i \in \mathcal{S}^-_{k+1}}$ to vertex $k+1$.
5) A $-l_{k+1}$-weight edge from each vertex $\{i\}_{i \in \mathcal{T}^+_{k+1}}$ to vertex $k+1$.
6) A $|l_i| - l_{k+1}$-weight edge from each vertex $\{i\}_{i \in \mathcal{T}^-_{k+1}}$ to vertex $k+1$.

The weight of the longest path from the START vertex to vertex $k+1$ is then as follows:

$$\begin{aligned} w_{k+1} &= \max\{0, \{w_i + l_i\}_{i \in \mathcal{S}^+_{k+1}}, \{w_i\}_{i \in \mathcal{S}^-_{k+1}}, \{w_i - l_{k+1}\}_{i \in \mathcal{T}^+_{k+1}}, \{w_i + |l_i| - l_{k+1}\}_{i \in \mathcal{T}^-_{k+1}}\} \\ &= \max\{0, \{\tau_i + l_i\}_{i \in \mathcal{S}^+_{k+1}}, \{\sigma_i\}_{i \in \mathcal{S}^-_{k+1}}, \{\tau_i - l_{k+1}\}_{i \in \mathcal{T}^+_{k+1}}, \{\sigma_i + |l_i| - l_{k+1}\}_{i \in \mathcal{T}^-_{k+1}}\}. \end{aligned} \tag{38}$$

The following relation follows by applying (36) and (38) when $l_{k+1} \geq 0$:

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

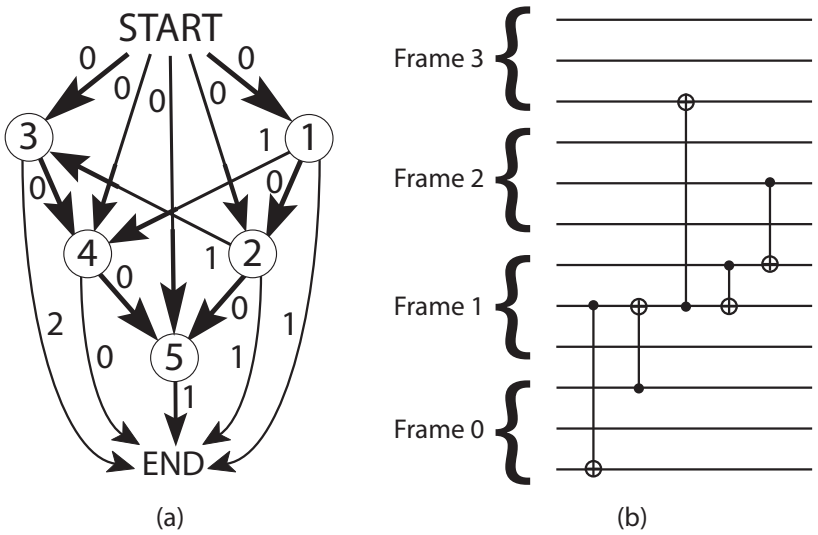

Fig. 9. (a) The commutativity graph G^+ and (b) a minimal-memory convolutional encoder for Example 3.

In a similar way, we can prove that

$$w_{k+1} = \sigma_{k+1},$$

when $l_{k+1} < 0$ and this last step concludes the proof. □

The complexity of constructing the graph G is $O(N^2)$ (the argument is similar to before), and dynamic programming finds the longest path in G in time linear in the number of its vertices and edges because G is a weighted, directed acyclic graph.

3.4.2 Example of a pearl-necklace encoder with arbitrary CNOT gates

We conclude the final development with an example.

Example 3. Consider the following succession of gate strings in a pearl-necklace encoder:

$$\overline{\text{CNOT}}(2,3)(D)\ \overline{\text{CNOT}}(1,2)(D^{-1})\ \overline{\text{CNOT}}(2,3)(D^{-2})\ \overline{\text{CNOT}}(1,2)(1)\ \overline{\text{CNOT}}(2,1)(D).$$

Figure 9(a) illustrates G for the above example. The longest path is

$$\text{START} \to 2 \to 3 \to \text{END}$$

with its weight equal to three. Thus, the minimal-memory convolutional encoder requires three frames of memory qubits. Also, from inspecting the graph G, we can determine the source qubit and target qubit frame indices in the convolutional encoder: $\tau_1 = 0$, $\sigma_2 = 0$, $\sigma_3 = 1$, $\tau = 1$, and $\tau_5 = 1$. Figure 9(b) depicts a minimal-memory convolutional encoder.

4 Conclusion

We have shown how to realize a minimal-memory convolutional encoder that implements the same transformation as a pearl-necklace encoder with arbitrary CNOT gate strings. Our approach is to construct a dependency graph whose directed edges represent non-commutative relations between gate strings in the pearl-necklace encoder. Determining the minimal memory is then the same task as determining the longest path through this graph. The algorithm for constructing and searching the graph requires time at worst quadratic in the number of gate strings in the pearl-necklace encoder. This technique should be useful when we have a pearl-necklace encoder description, which is the case in the work of Grassl and Rötteler [12], [15] and later work on entanglement-assisted quantum convolutional coding [17]. [16], [17], [18], [19], [20]

A later paper includes the general case of the algorithm for pearl-necklace encoders with gate strings other than CNOT gate strings [23]. The extension of the algorithm will include all gate strings that are in the shift-invariant Clifford group [12], [21], including Hadamard gates, phase gates, two variants of the controlled-phase gate string, and infinite-depth CNOT operations.

There might be ways to determine convolutional encoders with even lower memory requirements by using techniques different from those given here. First, our algorithm begins with a particular pearl-necklace encoder, i.e., a particular succession of gate strings to implement. One could first perform an optimization over all possible pearl-necklace encoders of a quantum convolutional code because there are many pearl-necklace encoders for a particular quantum convolutional code. Perhaps even better, one could look for a method to construct a repeated unitary directly from the polynomial description of the code itself. Ollivier and Tillich have considered such an approach in Sections 2.3 and 3 of Ref. [11], but it is not clear that their technique is attempting to minimize the memory resources for the encoder. There are well-developed techniques in the classical world to determine minimal-memory encoders. Ideally, we would like to have a "quantization" of Theorem 2.22 of Ref. [9].

The authors acknowledge useful discussions with Patrick Hayden and Pranab Sen. MMW acknowledges support from the MDEIE (Québec) PSR-SIIRI international collaboration grant.

References

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

[2] Peter Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium on the Foundations of Computer Science*, pages 124–134, 1994.

[3] Lov. K Grover. Quantum mechanics helps in searching for a needle in a haystack. *Physical Review Letters*, 79(2):325–328, 1997.

[4] Charles H. Bennett and Gilles Brassard. Quantum cryptography: Public key distribution and coin tossing. In *Proceedings of IEEE International Conference on Computers Systems and Signal*, pages 175–179, 1984.

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

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

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

[8] 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.

[9] Rolf Johannesson and Kamil Sh. Zigangirov. *Fundamentals of Convolutional Coding*. Wiley-IEEE Press, 1999.

[10] 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.

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

[12] 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.

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

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

[15] 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.

[16] 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.

[17] Mark M. Wilde and Todd A. Brun. Entanglement-assisted quantum convolutional coding. *Physical Review A*, 81(6):042333, April 2009.

[18] 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.

[19] 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.

[20] 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.

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

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

[23] Monireh Houshmand and Saied Hosseini Khayat. Minimal-memory realization of pearl-necklace encoders of general quantum convolutional codes. *arXiv:quant-ph/1009.2242*, 2010.

Monireh Houshmand was born in Mashhad, Iran. She got her B.S. and M.S. degrees in Electrical Engineering from Ferdowsi University of Mashhad, in 2005 and 2007, respectively. She is currently a Ph.D. student of electrical engineering in Ferdowsi University of Mashhad. Her research interests include quantum error correction and quantum cryptography.

Saied Hosseini Khayat (Member, IEEE) was born in Tehran, Iran. He received the BS degree in electrical engineering from Shiraz University, Iran (1986), and the MS and PhD degrees in electrical engineering from Washington University in St. Louis, USA (1991 and 1997). He held R&D positions at Globespan Semiconductor and Erlang Technologies. His expertise and interests lie in the area of digital system design for communications, networking and cryptography. At present, he is an assistant professor of electrical engineering at Ferdowsi University of Mashhad.

Mark M. Wilde (M'99) was born in Metairie, Louisiana. He obtained a B.S. in computer engineering from Texas A&M University in 2002, an M.S. in electrical engineering from Tulane University in 2004, and the Ph.D. in electrical engineering from the University of Southern California in 2008. Presently, he is a postdoctoral fellow in the School of Computer Science at McGill University in Montreal. His current research interests are in quantum Shannon theory and quantum error correction.