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

(Received 2 November 2010; published 10 February 2011)

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 algorithm's theoretical significance as a neat way of representing quantum convolutional codes, it is 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 to find a minimal-memory realization of a pearl-necklace encoder for Calderbank-Shor-Steane (CSS) convolutional codes. This work is an extension of 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 noncommutative 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.

DOI: 10.1103/PhysRevA.83.022308 PACS number(s): 03.67.Pp

I. INTRODUCTION

Quantum error correction codes are used to protect quantum information from decoherence and operational errors [1–10]. 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 [11–24] encodes an incoming stream of quantum information into an outgoing stream. Fast decoding algorithms exist for quantum convolutional codes [25].

The encoder for a quantum convolutional code has a representation as either a *standard encoder*[1] or a *pearl-necklace encoder*. The standard encoder [11,12,25] consists of a single unitary operator repeatedly applied to a stream of quantum data [see Fig. 1(a)]. On the other hand, the pearl-necklace encoder [see Fig. 1(b)] consists of several strings of the same unitary operator applied to the quantum data stream. Grassl and Rötteler [13] proposed an algorithm for encoding any quantum convolutional code with a pearl-necklace encoder. 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 in the 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 a quantum system.

It is trivial to determine the amount of memory required for implementation of a standard encoder: it is equal to the number of qubits that are fed back into the next iteration of the unitary operator that acts on the stream. For example, the standard encoders in Figs. 1(a), 2(c), and 5(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 its gates in the pearl-necklace encoder so that it becomes a standard encoder. Reference [27] was the first work that showed how standard encoders realize the transformations in the pearl-necklace encoders. Then in [26] we proposed an algorithm for finding the minimal-memory realization of a pearl-necklace encoder for the Calderbank-Shor-Steane (CSS) class of convolutional codes. This kind of encoder consists of controlled-NOT (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 [13]: Hadamard gates, phase gates, controlled-phase gate strings, and finite-depth and infinite-depth [22,24] CNOT operations. We show that there are many realizations for a given pearl-necklace encoder which are obtained considering noncommutativity relations of gate strings in the pearl-necklace encoder. Then for finding the minimal-memory realization, a specific graph, called a commutativity graph, is introduced. Each vertex in the 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

[*]monirehhoushmand@gmail.com
[†]saied.hosseini@gmail.com

[1]In previous literature including [26], this kind of encoder was simply called a "convolutional encoder."

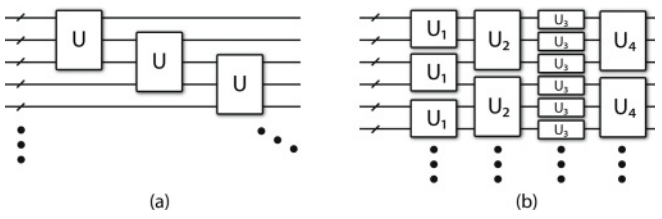

FIG. 1. Two different representations of the encoder for a quantum convolutional code: (a) a standard encoder and (b) a pearl-necklace encoder [26].

do not commute. The weight of a directed edge depends on the degrees of the two corresponding gate strings and their type of noncommutativity. 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 Sec. II we introduce some definitions and notation that are used in the rest of paper. In Sec. III, we first define three different types of noncommutativity and then propose an algorithm to find the minimal-memory requirements in a general case. Section IV concludes the paper.

II. DEFINITIONS AND NOTATION

We first provide some definitions and notation which are useful in our analysis later on. The gate strings in the pearl-necklace encoder and the gates in the standard encoder are numbered from left to right. We call the ith gate string in the pearl-necklace encoder, $\overline{U}_i$, and the ith gate in the standard 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 a CNOT or controlled-PHASE (CPHASE) gate. The notation $\overline{U}(a,bD^l)$ 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.[2]

Let U be either a 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 Fig. 2(a) correspond to

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

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

[2]Instead of previously used notation $\overline{U}(a,b)\,(D^l)$, we preferred to use the notation $\overline{U}(a,bD^l)$ because we think it better matches the concept.

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:

$$\begin{aligned} I^+_{\text{CNOT}} &= \{i\,|\,\overline{U}_i \text{ is } \overline{\text{CNOT}},\ l_i \geqslant 0,\ i \in \{1,2,\ldots,N\}\},\\ I^-_{\text{CNOT}} &= \{i\,|\,\overline{U}_i \text{ is } \overline{\text{CNOT}},\ l_i < 0,\ i \in \{1,2,\ldots,N\}\},\\ I^+_{\text{CPHASE}} &= \{i\,|\,\overline{U}_i \text{ is } \overline{\text{CPHASE}},\ l_i \geqslant 0,\ i \in \{1,2,\ldots,N\}\},\\ I^-_{\text{CPHASE}} &= \{i\,|\,\overline{U}_i \text{ is } \overline{\text{CPHASE}},\ l_i < 0,\ i \in \{1,2,\ldots,N\}\}. \end{aligned}$$

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:

$$\begin{aligned} I_H &= \{i\,|\,\overline{U}_i \text{ is } \overline{H},\ i \in \{1,2,\ldots,N\}\},\\ I_P &= \{i\,|\,\overline{U}_i \text{ is } \overline{P},\ i \in \{1,2,\ldots,N\}\}. \end{aligned}$$

Our convention for numbering the frames upon which the unitary operator of a standard encoder acts is from "bottom" to "top." Figure 6(b) illustrates this convention for a standard 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 standard encoder. If U_i is a Hadamard or phase gate, let τ_i denote the frame index of the target qubit of the U_i gate in a standard encoder. For example, consider the standard encoder in Fig. 6(b). The standard 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 standard encoder, the following notation is defined as follows:

CNOT $(a,b)\,(\sigma,\tau)$ denotes a CNOT gate from qubit a in frame σ to qubit b in frame τ.

CPHASE $(a,b)\,(\sigma,\tau)$ denotes a CPHASE gate from qubit a in frame σ to qubit b in frame τ.

$H(b)\,(\tau)$ denotes a Hadamard gate which acts on qubit b in frame τ.

$P(b)\,(\tau)$ denotes a phase gate which acts on qubit b in frame τ.

For example, the gates in Fig. 6(b) correspond to

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

III. MEMORY REQUIREMENTS FOR AN ARBITRARY PEARL-NECKLACE ENCODER

As mentioned in the Introduction, to find a practical realization of an arbitrary pearl-necklace encoder, it is required to rearrange its gates in the form of a standard encoder. The key idea behind this rearrangement is the same as the one in [26] and is as follows: One needs to find a set of gates consisting of a single gate from each gate string in the pearl-necklace encoder such that all remaining gates below the set commute with the set. This allows the remaining gates to be shifted to the right. Conceptually, this operation is repeated infinitely many times on the remaining gates to obtain a standard encoder.

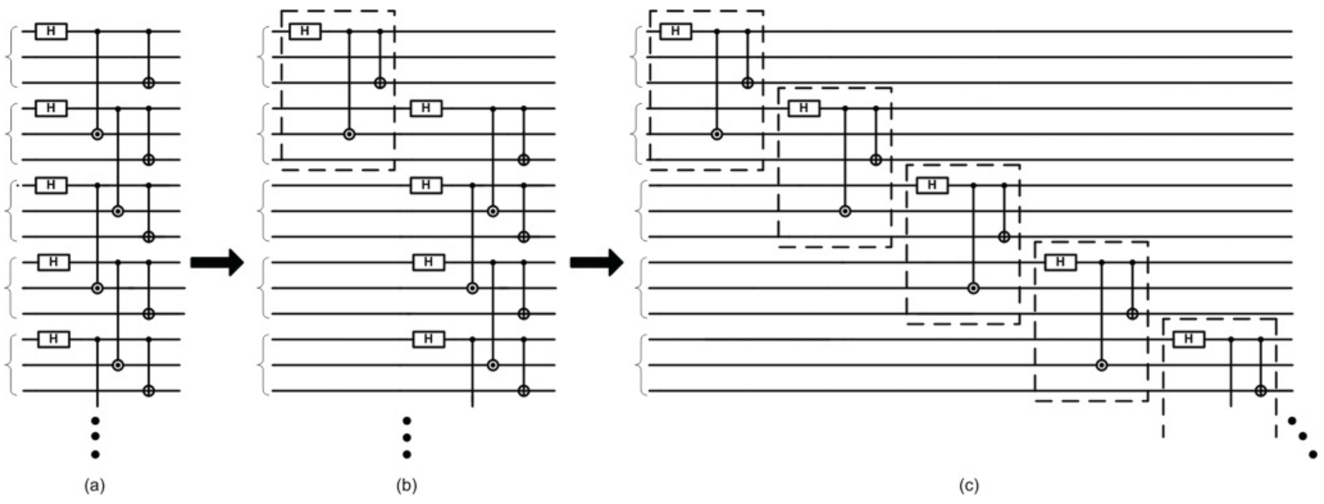

FIG. 2. Simple (since all gate strings commute with each other) example of the rearrangement of a pearl-necklace encoder for a non-CSS code into a standard encoder. (a) The pearl-necklace encoder consists of the gate strings $\overline{H}\,(1)\,\overline{\text{CPHASE}}\,(1,2)\,(D)\,\overline{\text{CNOT}}\,(1,3)$. (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 standard encoder from a pearl-necklace encoder.

In the case when all of the gates in the pearl-necklace encoder commute with each other, there will be no constraint on frame indices of the target qubits of the gates in the standard encoder (i.e., one is allowed to choose any gate from each gate string in the pearl-necklace encoder to build a standard encoder). Figure 2 shows an example of the rearrangement of commuting gate strings, $\overline{H}(1)\,\overline{\text{CPHASE}}\,(1,2)(D)\,\overline{\text{CNOT}}\,(1,3)$ to make a standard encoder. As shown in Fig. 2(b), the first gate from each of the three gate strings have been chosen as a set, and the remaining gates below this set have been shifted to the right. A repeated application of the same procedure on the remaining gates results in the standard encoder shown in Fig. 2(b).

On the other hand, we have shown in [26] that when the gate strings do not commute, the commutativity constraint of the remaining gates with the chosen set results in constraints on frame indices of target qubits of the standard encoder gates. This constraints are expressed in the form of three inequalities [Eqs. (6), (13), and (24)].

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

A. Different types of noncommutativity and their imposed constraints

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

1. Source-target noncommutativity

The gate strings in Eqs. (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 noncommutativity *source-target noncommutativity*.

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

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

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

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

With an analysis similar to the one in Sec. III A of [26] (whose underlying idea is similar to the one mentioned in the beginning of Sec. III), it can be proved that the following inequality applies to any correct choice of a standard encoder that implements either of the transformations in Eqs. (2)–(5):

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

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 standard encoder, respectively. We call the inequality in Eq. (6), *source-target constraint.*

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

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

In this case, $\sigma = 0 \leqslant \tau' = 0$. Since the source-target constraint is satisfied, the remaining gates after the chosen set [the highlighted gates in Fig. 3(a)] can be shifted to the right [Fig. 3(b)]. Repeated application of the procedure in Fig. 3(b) realizes a standard encoder representation from a pearl-necklace encoder [Fig. 3(c)]. Figure 4(a) shows the same gate strings as Fig. 3(a). In this case the chosen standard encoder is

$$\text{CPHASE}(2,3)\,(1,1)\text{CNOT}(1,2)\,(1,0).\tag{8}$$

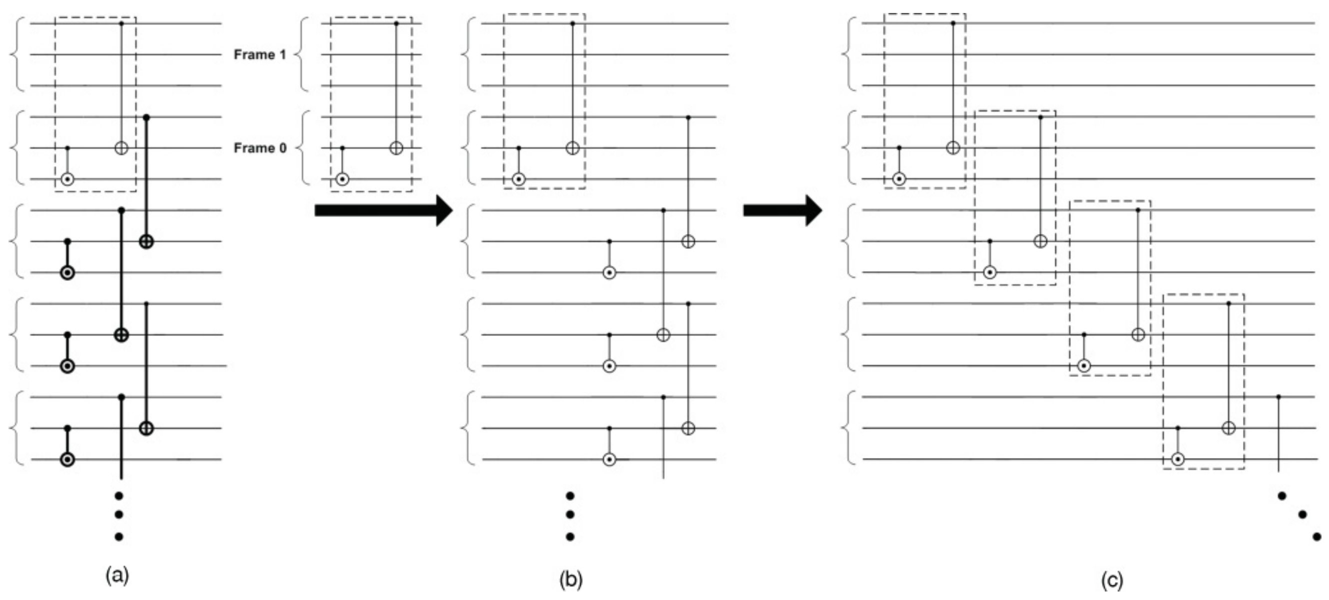

FIG. 3. Finding a correct standard encoder for a two noncommutative gate strings. (a) The pearl-necklace encoder consists of the gate strings $\overline{\text{CPHASE}}(2,3D)\overline{\text{CNOT}}(1,2D)$, which have source-target noncommutativity. (b) The rearrangement of gates after the first three by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

Here, $\sigma = 1 \nleqslant \tau' = 0$. Figure 4(b) shows that one of the gates that remains after the chosen set does not commute with it. Therefore CPHASE $(2,3)\,(1,1)$ CNOT $(1,2)\,(1,0)$ is not a correct choice for the standard encoder.

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

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

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 noncommutativity and returns FALSE otherwise.

2. *Target-source noncommutativity*

It is obvious that the gate strings in Eqs. (9)–(12) 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 noncommutativity, *target-source noncommutativity*.

$$\overline{\text{CNOT}}\,(a,bD^{l})\,\overline{\text{CNOT}}\,(a',b'D^{l'}),\quad \text{where}\quad b=a',\tag{9}$$
$$\overline{\text{CNOT}}\,(a,bD^{l})\,\overline{\text{CPHASE}}\,(a',b'D^{l'}),\quad \text{where}\quad b=a',\tag{10}$$
$$\overline{H}(b)\,\overline{\text{CNOT}}\,(a',b'D^{l'}),\quad \text{where}\quad b=a',\tag{11}$$
$$\overline{H}(b)\,\overline{\text{CPHASE}}\,(a',b'D^{l'}),\quad \text{where}\quad b=a'.\tag{12}$$

With an analysis similar to the analysis in Sec. III A of [26] (whose underlying idea is similar to the one mentioned in the beginning of Sec. III), it can be proved that the following inequality applies to any correct choice of a standard encoder that implements either of the transformations in Eqs. (9)–(12):

$$\tau \leqslant \sigma',\tag{13}$$

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 standard encoder respectively. We call the inequality in Eq. (13), *target-source constraint.*

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

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

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 noncommutativity and returns FALSE otherwise.

3. *Target-target noncommutativity*

It is obvious that the gate strings in Eqs. (14)–(23) 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 noncommutativity, *target-target noncommutativity*.

$$\overline{\text{CPHASE}}\,(a,bD^{l})\,\overline{\text{CNOT}}\,(a',b'D^{l'}),\quad \text{where}\quad b=b',\tag{14}$$
$$\overline{\text{CNOT}}\,(a,bD^{l})\,\overline{\text{CPHASE}}\,(a',b'D^{l'}),\quad \text{where}\quad b=b',\tag{15}$$
$$\overline{\text{CNOT}}\,(a,bD^{l})\,\overline{H}(b),\quad \text{where}\quad b=b',\tag{16}$$
$$\overline{\text{CPHASE}}\,(a,bD^{l})\,\overline{H}(b'),\quad \text{where}\quad b=b',\tag{17}$$
$$\overline{H}(b)\,\overline{\text{CNOT}}\,(a',b'D^{l'}),\quad \text{where}\quad b=b',\tag{18}$$
$$\overline{H}(b)\,\overline{\text{CPHASE}}\,(a',b'D^{l'}),\quad \text{where}\quad b=b',\tag{19}$$
$$\overline{\text{CNOT}}\,(a,bD^{l})\,\overline{P}(b'),\quad \text{where}\quad b=b',\tag{20}$$
$$\overline{P}(b)\,\overline{\text{CNOT}}\,(a',b'D^{l'}),\quad \text{where}\quad b=b',\tag{21}$$
$$\overline{P}(b)\overline{H}(b'),\quad \text{where}\quad b=b',\tag{22}$$
$$\overline{H}(b)\overline{P}(b'),\quad \text{where}\quad b=b'.\tag{23}$$

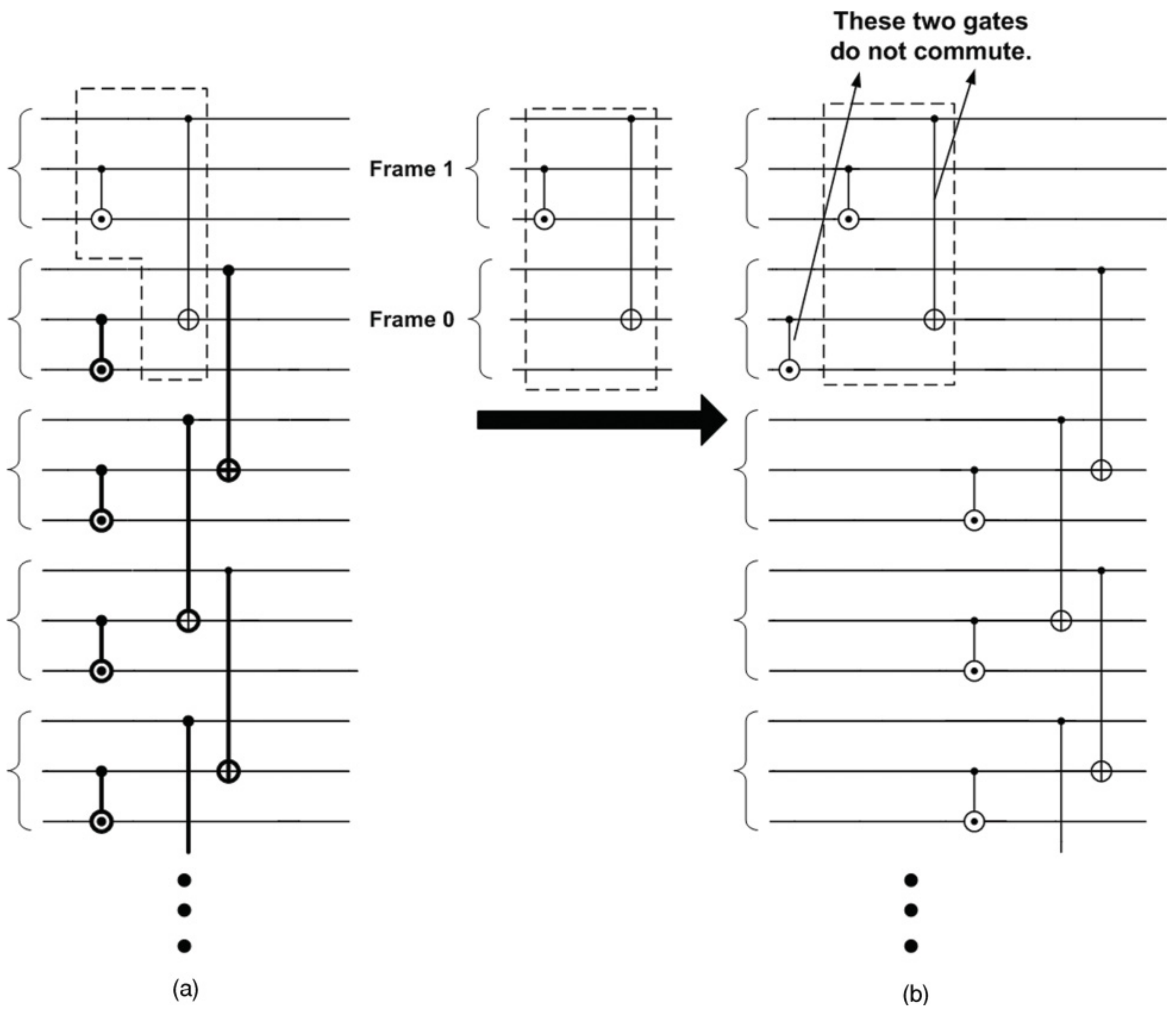

FIG. 4. (a) An incorrect standard encoder for the gate strings in Fig. 3. (b) Since the source-target constraint is not satisfied, one of the gates that remains after the chosen set does not commute with it.

With a analysis similar to the analysis in Sec. III A of [26] (whose underlying idea is similar to the one mentioned in the beginning of Sec. III), it can be proved that the following inequality applies to any correct choice of a standard encoder that implements either of the transformations in Eqs. (14)–(23):

$$\tau \leqslant \tau', \tag{24}$$

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 standard encoder, respectively. We call the inequality in Eq. (24), *target-target constraint.* The following Boolean function is used to determine whether target-target noncommutativity exists for two gate strings:

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

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 noncommutativity and returns FALSE otherwise.

Consider the jth 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 noncommutativity. Therefore we define the following sets:

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

B. 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 includes all gate strings that are in the shift-invariant Clifford group: Hadamard gates, phase gates, CPHASE, and finite-depth and infinite-depth CNOT gate strings. To achieve this goal, we consider any noncommutativity 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:

$$\overline{U_1},\overline{U_2},\ldots,\overline{U_N}. \tag{25}$$

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

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

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

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

If the first gate string is $\overline{\text{CPHASE}}\,(a_1,b_1D^{l_1}),l_1 \geqslant 0$, the first gate in the standard encoder is

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

If the first gate string is $\overline{\text{CPHASE}}\,(a_1,b_1D^{l_1}),l_1 < 0$, the first gate in the standard encoder is

$$\text{CPHASE}\,(a_1,b_1)\,(\sigma_1 = 0,\tau_1 = |l_1|). \tag{29}$$

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

$$H(b_1)\,(0), \tag{30}$$
$$P(b_1)\,(0). \tag{31}$$

For the target indices of each gate j where $2 \leqslant j \leqslant N$, 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 Eq. (6) we have

$$\begin{gathered}\sigma_i \leqslant \tau_j \ \ \forall i \in (\mathcal{S}-\mathcal{T})_j,\\ \therefore\ \tau_i + l_i \leqslant \tau_j \ \ \forall i \in (\mathcal{S}-\mathcal{T})_j,\\ \therefore\ \max\{\tau_i + l_i\}_{i\in(\mathcal{S}-\mathcal{T})_j} \leqslant \tau_j.\end{gathered} \tag{32}$$

By applying the target-source constraint in Eq. (13) we have

$$\begin{gathered}\tau_i \leqslant \sigma_j \ \ \forall i \in (\mathcal{T}-\mathcal{S})_j,\\ \therefore\ \tau_i \leqslant \tau_j + l_j \ \ \forall i \in (\mathcal{T}-\mathcal{S})_j,\\ \therefore\ \tau_i - l_j \leqslant \tau_j \ \ \forall i \in (\mathcal{T}-\mathcal{S})_j,\\ \therefore\ \max\{\tau_i - l_j\}_{i\in(\mathcal{T}-\mathcal{S})_j} \leqslant \tau_j.\end{gathered} \tag{33}$$

By applying the target-target constraint in Eq. (24) we have

$$\tau_i \leqslant \tau_j \ \ \forall i \in (\mathcal{T}-\mathcal{T})_j, \quad \therefore\ \max\{\tau_i\}_{i\in(\mathcal{T}-\mathcal{T})_j} \leqslant \tau_j. \tag{34}$$

The following constraint applies to the frame index τ_j of the target qubit by applying Eqs. (32)–(34):

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

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

$$\tau_j = \max\left\{\{\tau_i + l_i\}_{i\in(\mathcal{S}-\mathcal{T})_j},\{\tau_i - l_j\}_{i\in(\mathcal{T}-\mathcal{S})_j},\{\tau_i\}_{i\in(\mathcal{T}-\mathcal{T})_j}\right\}. \tag{36}$$

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 $l_j \geqslant 0$ we choose the frame index τ_j as follows:

$$\tau_j = 0. \tag{37}$$

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

$$\tau_j = |l_j|. \tag{38}$$

If $l_j \geqslant 0$, a good choice for the frame index τ_j, by considering Eqs. (36) and (37), is as follows:

$$\tau_j = \max\left\{0,\{\tau_i + l_i\}_{i\in(\mathcal{S}-\mathcal{T})_j},\{\tau_i - l_j\}_{i\in(\mathcal{T}-\mathcal{S})_j},\{\tau_i\}_{i\in(\mathcal{T}-\mathcal{T})_j}\right\}. \tag{39}$$

And if $l_j < 0$, a good choice for the frame index τ_j, by considering Eqs. (36) and (38), is as follows:

$$\tau_j = \max\left\{|l_j|,\{\tau_i + l_i\}_{i\in(\mathcal{S}-\mathcal{T})_j},\{\tau_i - l_j\}_{i\in(\mathcal{T}-\mathcal{S})_j},\{\tau_i\}_{i\in(\mathcal{T}-\mathcal{T})_j}\right\}. \tag{40}$$

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 Eq. (6) we have

$$\begin{gathered}\sigma_i \leqslant \tau_j \ \ \forall i \in (\mathcal{S}-\mathcal{T})_j,\\ \therefore\ \tau_i + l_i \leqslant \tau_j \ \ \forall i \in (\mathcal{S}-\mathcal{T})_j,\\ \therefore\ \max\{\tau_i + l_i\}_{i\in(\mathcal{S}-\mathcal{T})_j} \leqslant \tau_j.\end{gathered} \tag{41}$$

By applying target-target constraint in Eq. (24) we have

$$\tau_i \leqslant \tau_j \ \ \forall i \in (\mathcal{S}-\mathcal{T})_j, \quad \therefore\ \max\{\tau_i\}_{i\in(\mathcal{T}-\mathcal{T})_j} \leqslant \tau_j, \tag{42}$$

The following constraint applies to the frame index τ_j of the target qubit by applying Eqs. (41) and (42):

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

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

$$\tau_j = \max\left\{\{\tau_i + l_i\}_{i\in(\mathcal{S}-\mathcal{T})_j},\{\tau_i\}_{i\in(\mathcal{T}-\mathcal{T})_j}\right\}. \tag{43}$$

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:

$$\tau_j = 0. \tag{44}$$

A good choice for the frame index τ_j, by considering Eqs. (43) and (44), is

$$\tau_j = \max\left\{0,\{\tau_i + l_i\}_{i\in(\mathcal{S}-\mathcal{T})_j},\{\tau_i\}_{i\in(\mathcal{T}-\mathcal{T})_j}\right\}. \tag{45}$$

Now consider $\overline{U_j}$ is the $\overline{P}$, then by applying the target-target constraint in Eq. (24), the following inequality must be satisfied to target index of $\overline{U_j}$, τ_j:

$$\tau_i \leqslant \tau_j \ \ \forall i \in (\mathcal{T}-\mathcal{T})_j, \quad \therefore \ \max\{\tau_i\}_{i\in(\mathcal{T}-\mathcal{T})_j} \leqslant \tau_j.$$

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

$$\tau_j = \max\{\tau_i\}_{i\in(\mathcal{T}-\mathcal{T})_j}. \tag{46}$$

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:

$$\tau_j = 0. \tag{47}$$

A good choice for the frame index τ_j, by considering Eqs. (46) and (47), is

$$\tau_j = \max\left\{0, \{\tau_i\}_{i\in(\mathcal{T}-\mathcal{T})_j}\right\}. \tag{48}$$

1. Construction of the commutativity graph

We introduce the notion of a *commutativity* 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 noncommutativity relations of the gate strings. $\mathcal{G}$ consists of N vertices, labeled $1,2,\ldots,N$, where the jth vertex corresponds to the jth 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.

If the degree of the jth gate string is negative, a $|l_j|$-weight edge connects the START vertex to vertex j; otherwise a zero-weight edge connects the START vertex to vertex j. If the degree of the jth gate string is positive, an l_j-weight edge connects vertex j to the END vertex; otherwise, a zero-weight edge connects vertex j to the END vertex. Two vertices i and j are connected to each other if ith gate string and jth gate string do not commute. The weight of the edge depends on the degrees of the two corresponding gate strings and their type of noncommutativity.

The 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 in Eq. (25). 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 instruction in the inner **for** loop requires constant time $O(1)$, the sum of iterations of the **if** instruction in the jth iteration of the outer **for** loop is equal to $j-1$. Thus the running time $T(N)$ of Algorithm 1 is

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

Algorithm 1 presents the pseudocode for constructing the commutativity graph.

```
$N \leftarrow$ 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 $j^{\text{th}}$ gate string $\overline{U}_j$
  if $j \in (I^-_{\text{CNOT}} \cup I^-_{\text{CPHASE}})$ then
    DrawEdge(**START**, $j$, $|l_j|$)
  else
    DrawEdge(**START**, $j$, 0)
  end if
  for $i := 1$ to $j-1$ do
    if $j \in (I^+_{\text{CNOT}} \cup I^+_{\text{CPHASE}} \cup I^-_{\text{CNOT}} \cup I^-_{\text{CPHASE}})$ then
      if $i \in (\mathcal{S}-\mathcal{T})_j$ then
        DrawEdge($i, j, l_i$ )
      end if
      if $i \in (\mathcal{T}-\mathcal{S})_j$ then
        DrawEdge($i, j, -l_j$)
      end if
      if $i \in (\mathcal{T}-\mathcal{T})_j$ then
        DrawEdge($i, j, 0$)
      end if
    else
      if $j \in I_H$ then
        if $i \in (\mathcal{S}-\mathcal{T})_j$ then
          DrawEdge($i, j, l_i$)
        end if
        if $i \in (\mathcal{T}-\mathcal{T})_j$ then
          DrawEdge($i, j, 0$)
        end if
      else
        if $i \in (\mathcal{T}-\mathcal{T})_j$ 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 $j \in (I^+_{\text{CNOT}} \cup I^+_{\text{CPHASE}})$ then
    DrawEdge($j$,**END**, $l_j$)
  else
    DrawEdge($j$,**END**, 0)
  end if
end for
```

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 standard encoder implementation.

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

$$w_j = \tau_j. \tag{49}$$

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 Eqs. (26), (28), (30), and (31), $\tau_1 = 0$, 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 Eqs. (27) and (29), $\tau_1 = |l_1|$, therefore $w_1 = \tau_1 = |l_1|$. Thus the base step holds for the target index of the first gate in a minimal-memory standard encoder. Now suppose the property holds for the target indices of the first k gates in the standard encoder:

$$w_j = \tau_j \;\; \forall j : 1 \leqslant j \leqslant k.$$

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

$$\begin{aligned} w_{k+1} &= \max\left\{0,\{w_i+l_i\}_{i\in(\mathcal{S}-\mathcal{T})_{k+1}},\{w_i-l_{k+1}\}_{i\in(\mathcal{T}-\mathcal{S})_{k+1}},\{w_i\}_{i\in(\mathcal{T}-\mathcal{T})_{k+1}}\right\},\\ &= \max\left\{0,\{\tau_i+l_i\}_{i\in(\mathcal{S}-\mathcal{T})_{k+1}},\{\tau_i-l_{k+1}\}_{i\in(\mathcal{T}-\mathcal{S})_{k+1}},\{\tau_i\}_{i\in(\mathcal{T}-\mathcal{T})_{k+1}}\right\}. \end{aligned} \tag{50}$$

By applying Eqs. (39) and (50) 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, then

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

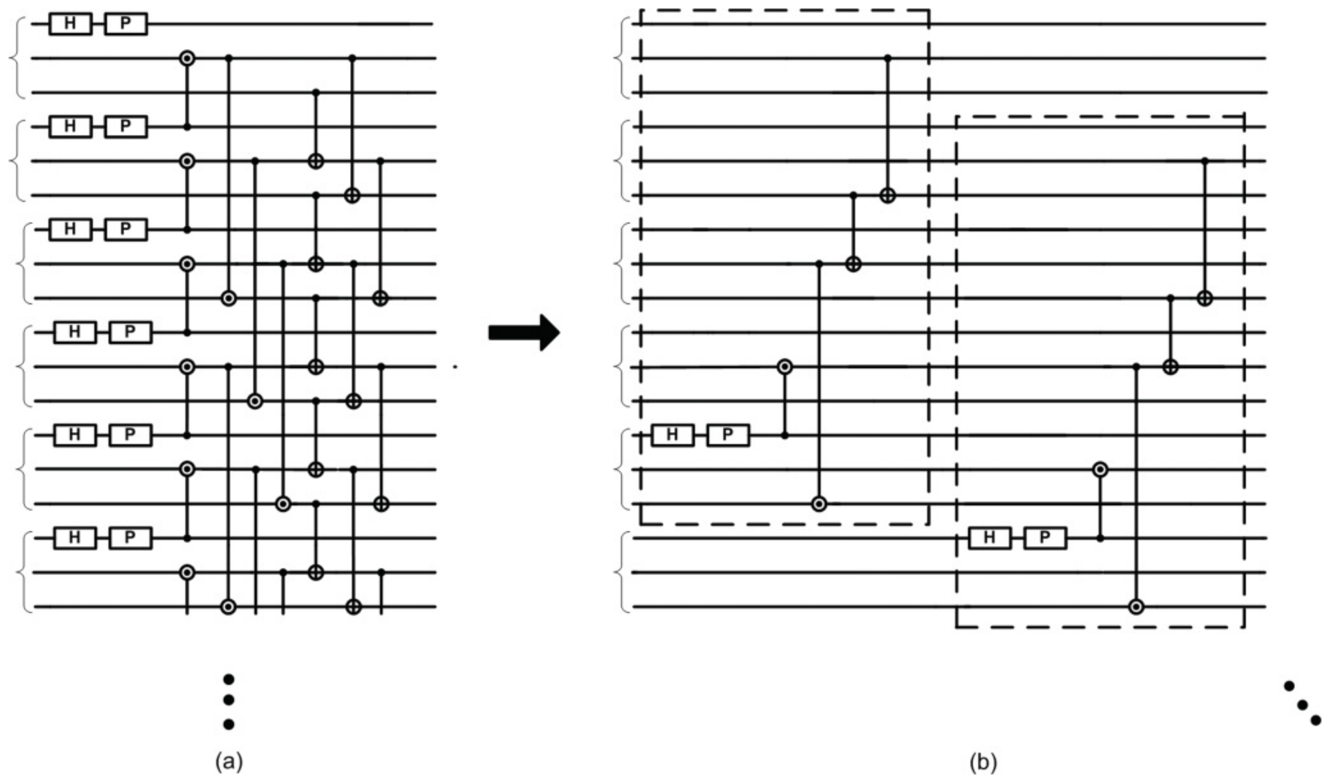

FIG. 5. (a) Pearl-necklace representation, and (b) minimal-memory standard encoder representation for example 1.

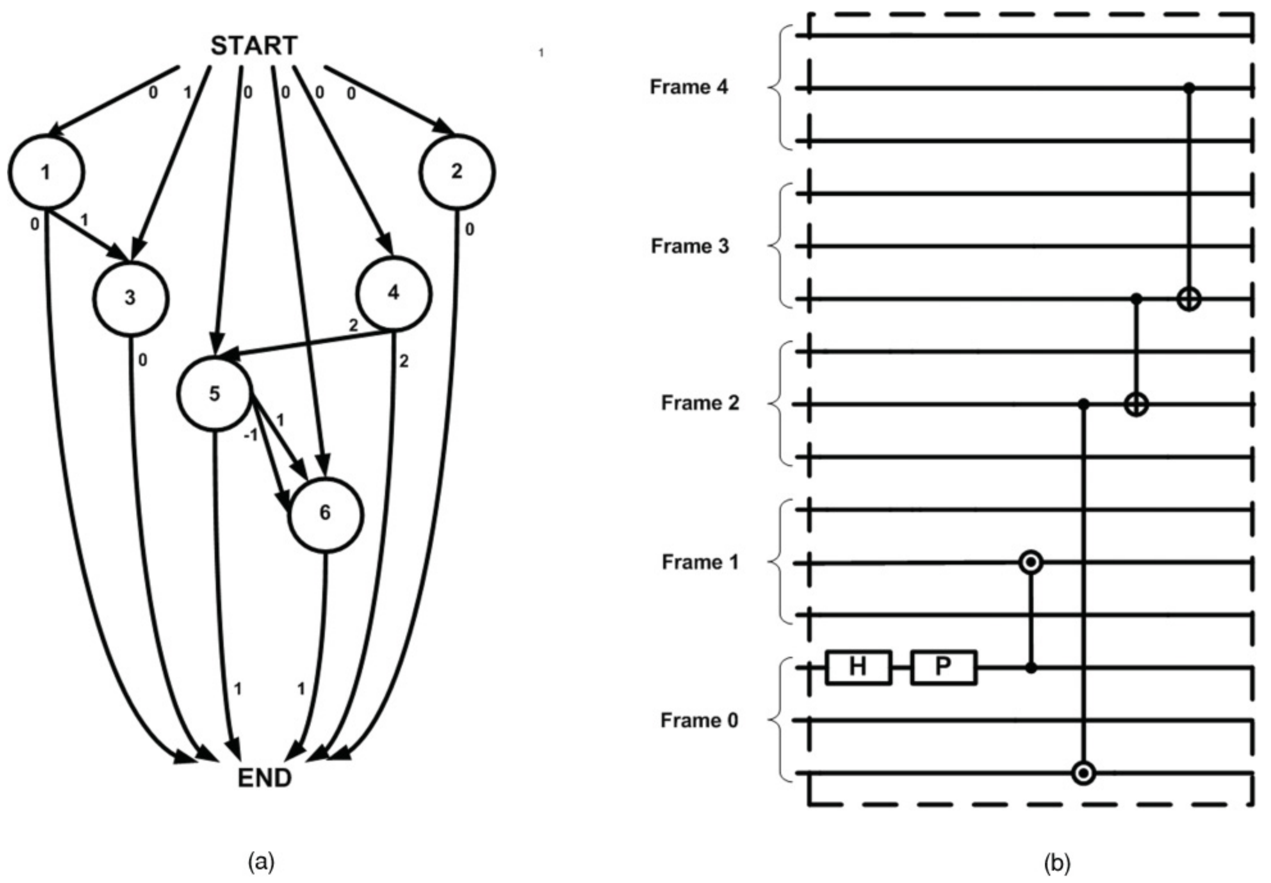

FIG. 6. (a) Commutativity graph $\mathcal{G}$ and (b) a minimal-memory standard encoder for example 1.

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

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

The first equality holds because the longest path in the graph is the maximum of the weight of the path to the ith vertex summed with the weight of the edge to the END vertex. The second equality follows by applying Eq. (49). The final equality follows because $\sigma_i = \tau_i + l_i$. It is clear that

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

is equal to minimal required memory for a minimal-memory standard 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 a nondeterministic polynomial time-complete (NP-complete) problem; but in a weighted, directed acyclic graph, it requires linear time with dynamic programming [28].

Example 1. Consider the following succession of gate strings in a pearl-necklace encoder [Fig. 5(a)]:

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

Figure 6(a) draws $\mathcal{G}$ for this pearl-necklace encoder, after running the algorithm. The longest path through the graph is

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

with weight equal to four $(0+2+1+1)$. Therefore the minimal memory for the standard 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: $\tau_1 = 0,\quad \tau_2 = 0,\quad \tau_3 = 1,\quad \tau_4 = 0,\quad \tau_5 = 2,\quad$ and $\quad \tau_6 = 3$. Figure 6(b) depicts a minimal-memory standard encoder for this example. Figure 5(b) depicts minimal-memory standard encoder representation for the pearl-necklace encoder in Fig. 5(a).

IV. CONCLUSION

In this paper, we have proposed an algorithm to find a practical realization with a minimal-memory requirement for 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 noncommutativity relations of gate strings in the encoder determine the realization. We introduce a commutativity graph, whose vertices each correspond to a gate string in the pearl-necklace encoder. The weighted edges represent noncommutativity 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 constructing the graph and finding the longest path is quadratic in the number of gate strings in the pearl-necklace encoder.

As mentioned in our previous paper [26], there is an open question remaining. Our proposed algorithm begins with a given pearl-necklace encoder of a general quantum convolutional code, and determines its minimal-memory standard encoder. There exist, however, many pearl-necklace encoders corresponding to a given quantum convolutional code. Our algorithm does not necessarily find a minimal-memory encoder among all possible realizations of the code.

An open problem is to determine a minimal-memory standard encoder for a general quantum convolutional code that is described by its polynomial matrix. There are two possible approaches to solving this problem: (1) to determine the pearl-necklace encoder that requires minimum memory among all pearl-necklace encoders realizing the same code, and (2) to construct a standard quantum convolutional encoder directly from a given polynomial matrix description in such a way that the required memory is minimized.

ACKNOWLEDGMENT

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

[1] F. Gaitan, *Quantum Error Correction and Fault Tolerant Quantum Computing* (CRC Press, Taylor and Francis Group, 2008).
[2] D. A. Lidar, I. L. Chuang, and K. B. Whaley, Phys. Rev. Lett. **81**, 2594 (1998).
[3] D. Gottesman, Ph.D. thesis, California Institute of Technology, 1997, e-print arXiv:quant-ph/9705052.
[4] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information* (Cambridge University Press, 2000).
[5] A. M. Steane, Phys. Rev. Lett. **77**, 793 (1996).
[6] A. R. Calderbank and P. W. Shor, Phys. Rev. A **54**, 1098 (1996).
[7] A. R. Calderbank, E. M. Rains, P. W. Shor, and N. J. A. Sloane, Phys. Rev. Lett. **78**, 405 (1997).
[8] A. R. Calderbank, E. M. Rains, P. W. Shor, and N. J. A. Sloane, IEEE Trans. Inf. Theory **44**, 1369 (1998).
[9] P. Zanardi and M. Rasetti, Phys. Rev. Lett. **79**, 3306 (1997).
[10] P. Zanardi and M. Rasetti, Mod. Phys. Lett. B **11**, 1085 (1997).
[11] H. Ollivier and J.-P. Tillich, Phys. Rev. Lett. **91**, 177902 (2003).
[12] H. Ollivier and J.-P. Tillich, e-print arXiv:quant-ph/0401134.
[13] M. Grassl and M. Rötteler, in *Proceedings of the 2006 IEEE International Symposium on Information Theory* (IEEE, New York, 2006), p. 1109.
[14] M. Grassl and M. Rötteler, in *Proceedings of the Forty-Fourth Annual Allerton Conference on Communication, Control, and Computing, 2006* (unpublished), p. 510.
[15] M. Grassl and M. Rötteler, in *Proceedings of the 2007 IEEE International Symposium on Information Theory* (IEEE, New York, 2007), p. 816.
[16] G. D. Forney and S. Guha, in *Proceedings of the 2005 IEEE International Symposium on Information Theory* (IEEE, New York, 2005), p. 1028.
[17] G. D. Forney, M. Grassl, and S. Guha, IEEE Trans. Inf. Theory **53**, 865 (2007).
[18] S. A. Aly, M. Grassl, A. Klappenecker, M. Rötteler, and P. K. Sarvepalli, in *10th Canadian Workshop on Information Theory, 2007* (unpublished), p. 180, e-print arXiv:quant-ph/0703113.
[19] S. A. Aly, A. Klappenecker, and P. K. Sarvepalli, in *Proceedings of the 2007 IEEE International Symposium on Information Theory* (IEEE, New York, 2007), p. 821.
[20] M. M. Wilde, H. Krovi, and T. A. Brun, in *Proceedings of the 2010 IEEE International Symposium on Information Theory* (IEEE, New York, 2010).
[21] M. M. Wilde and T. A. Brun, Phys. Rev. A **81**, 042333 (2010).
[22] M. M. Wilde and T. A. Brun, in *Proceedings of the 2008 IEEE International Symposium on Information Theory* (IEEE, New York, 2008), p. 359.
[23] M. M. Wilde and T. A. Brun, Quantum Inf. Process. **9**, 509 (2010).
[24] M. M. Wilde and T. A. Brun, Phys. Rev. A **79**, 032313 (2009).
[25] D. Poulin, J.-P. Tillich, and H. Ollivier, IEEE Trans. Inf. Theory **55**, 2776 (2009).
[26] M. Houshmand, S. Hosseini-Khayat, and M. M. Wilde, IEEE Trans. Comput. (to be published), e-print arXiv:1004.5179v2.
[27] M. M. Wilde, Phys. Rev. A **79**, 062325 (2009).
[28] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms* (MIT Press, 2009).