Architecture-Specific Mapping of CNOT Gates

Allen Mi

Draft: December 2018
First Revision: February–March 2019

Abstract

We study the problem of mapping quantum circuits to quantum computer architectures. Specifically, we propose and prove lower bounds on gate count and circuit depth for mapping CNOT gates to line graph architectures. We present two algorithms that provide 2-approximations of these bounds, and propose efficient generalizations to arbitrary architectures. Additionally, we discuss general heuristics on approaching this problem, and how they lead to the discovery of the two algorithms.

1 Introduction

The area of quantum computation has seen rapid advancement over recent years. An important measure that symbolizes this development is the ever-increasing qubit count of quantum computer implementations. Notable examples that are currently operational include the 5-qubit IBM Tenerife [7] and Yorktown [8], the 14/16-qubit IBM Melbourne [5], the 16-qubit IBM Rueschlikon [6], and the 72-qubit Google Bristlecone. While the growth in qubit count gives rise to more possibilities in running complex quantum programs on real hardware, it has brought forth a new problem: the connectivity between qubits. For instance, the 5-qubit IBM Tenerife adopts a "bow-tie" architecture [7], where the maximum undirected distance between two registers is 2; the 16-qubit IBM Rueschlikon, however, adopts a ladder graph architecture [6], where the maximum undirected distance between two registers is 8. The increased distance between registers has severe implications for the efficiency and accuracy of multi-qubit interactions, as more bi-qubit gates between neighboring registers must be employed to facilitate this interaction. For a quantum computer architecture that supports the Clifford+T gate set, the task of effectively supporting multi-qubit operations resolves to efficiently implementing CNOT gates between two non-neighboring registers.

This paper is focused on studying this problem from a combinatorial perspective. We introduce a few concepts and their formalisms in Section 2, accompanied by a brief overview of previous work on this topic. A formal definition

of the architecture-specific CNOT mapping problem is introduced in Section 3. Section 4 is devoted to proving lower bounds on special cases of the problem. We discuss the general heuristics for solving the special cases in Section 5, and present two efficient algorithms in Section 6. Efficient generalizations of these algorithms are provided in Section 7. Finally, Section 8 concludes this paper.

2 Background

In this section, we introduce and formalize a few essential concepts. We then provide a brief overview of previous work conducted on this topic. See Appendix A.1 and Appendix A.2 for notations specific to this text.

2.1 Formalization

Let C^n denote a *quantum circuit* on n qubits. We provide a formalization of C^n that would simplify the introduction and proof of the ideas in this paper. Let $C^n = (F_1, F_2, \ldots, F_d)$ be an ordered list of *frames*, with each frame $F_i = \{g_{i1}, g_{i2}, \ldots, g_{ij}\}$ be a set of quantum gates. Each g_{ik} represents a quantum gate that operates on one or more quantum registers. Let $\mathrm{reg}(g_{ik})$ denote an ordered list of quantum registers that g_{ik} operates on. For instance, if $g_{ik} = \mathrm{Toffoli}(r_1, r_2, r_3)$, then $\mathrm{reg}(g_{ik}) = (r_1, r_2, r_3)$. For any $F_i \in C^n$, it is required that $\forall g_{ik}, g_{il} \in F_i$, g_{ik} and g_{il} can be executed simultaneously. Formally, the requirement can be expressed as follows:

$$\forall F_i \in C^n, \forall g_{ik}, g_{il} \in F_i, \mathrm{reg}(g_{ik}) \cap \mathrm{reg}(g_{il}) = \varnothing. \tag{1}$$

In other words, the two gates g_{ik} and g_{il} can be placed within the same frame if the sets of gates they operate on are disjoint.

Under this formalization, the *circuit depth* of a quantum circuit C^n can be defined by the number of frames in C^n. For a quantum computer, all gates within the same frame can be executed simultaneously. Hence circuit depth is instrumental in representing the time complexity of a quantum circuit.

The *gate count* of a quantum circuit C^n is equal to the number of gates in all frames of C^n. Formally, this quantity is equal to $\sum_{F_i \in C^n} |F_i|$.

We now proceed to definitions related to hardware. The *architecture* of an n-bit quantum computer can be represented by a directed graph $G^n = (R, E)$, where each vertex $r \in R$ represents a quantum register, and each edge $(u, v) \in E$ represents a connection between quantum registers. The direction of each edge is essential for some implementations. For instance, when the connection is implemented via cross-resonance interaction, the interaction between the qubits is more pronounced when choosing the higher-frequency register as the control [7]. In this sense, each edge (u, v) in E corresponds to a possible location for bi-qubit gate placement, where u acts as the first register, and v acts as the second register. Since we base our discussions on the Clifford+T gate set, the bi-qubit gate of interest is CNOT. In this case, u is the controlling register, and v is the controlled register in the computational basis $\{|0\rangle, |1\rangle\}$. As per

convention, we only admit G^n as a valid n-qubit architecture if G^n is weakly connected.

As an aside, we define the *complete digraph architecture* G^n_* on n qubits by the complete digraph $(R, R \times R)$. In G^n_*, for an arbitrary pair of registers (r_1, r_2), both (r_1, r_2) and (r_2, r_1) are present in the set of connections.

We define the *architectural dependency* of a quantum circuit C^n as a directed graph $G^n_D = (R_D, E_D)$ where $|R_D| = n$, $E_D \subseteq R_D \times R_D$. Let $R_D = \{r_1, r_2, \ldots, r_n\}$, and we define E_D as the set of connections the CNOT gates in C^n require. Formally, E_D is defined as follows:

$$E_D = \{(r_i, r_j) \in R_D \times R_D \mid \exists F_k \in C^n \text{ s.t. } \mathrm{CNOT}(r_i, r_j) \in F_k\}. \tag{2}$$

Additionally, let C^n_* denote an n-qubit quantum circuit that has an architectural dependency G^n_*. The G^n_* model is most commonly used for developing and communicating quantum algorithms, as the complete digraph architecture enables simple expression of arbitrary CNOT gates.

Next, we define the notion of *mapping*. Let Γ^n denote the set of all quantum circuits on n qubits. We say an operation $M_{G^n} : \Gamma^n \to \Gamma^n$ is a valid mapping for circuit C^n with respect to some specific n-qubit architecture G^n iff the architectural dependency of $M_{G^n}(C^n)$ is a subgraph of G^n, and $M_{G^n}(C^n)$ and C^n produce identical output states on any input state $|\chi\rangle \coloneqq |q_1 q_2 \ldots q_n\rangle$. Metaphorically, a mapping can be considered a "compiling" process of C^n that targets a specific architecture G^n.

Finally, it is worth noting that without loss of generality, all discussions of CNOT gates in this text are performed on the computational basis $\{|0\rangle, |1\rangle\}$, except where Appendix A.3 is applied. Since the permutation matrix of the CNOT gate is not dependent on the eigenbases of its operands, the results derived apply to other bases as well.

2.2 Previous Work

While much study has been performed on quantum circuit synthesis, the interest about this specific topic only began very recently. In 2014, Wille et al. [10] reviewed existing approaches to quantum circuit synthesis, and proposed exact methods for reordering circuit lines by SWAP insertions. In 2017, Zulehner et al. [11] proposed an algorithm based on A^* search for exact reordering of qubits. In late 2017, Zulehner et al. [12] further proposed a heuristics-based mapping algorithm specific to IBM QX architectures. Additionally, IBM Q Team [1] maintains a mapping mechanism for QX architectures as part of the `qiskit` package.

As CNOT is the basic bi-qubit gate of the Clifford+T gate set, this paper directly targets the fundamental problem of mapping CNOT gates to arbitrary architectures.

3 Problem Statement

In the proceeding sections, we study how a CNOT gate between two quantum registers can be efficiently mapped with respect to an architecture. Given an arbitrary n-qubit architecture G^n as a weakly connected graph (R, E) s.t. $|R| = n$, where R denotes the set of quantum registers, and $E \subseteq R \times R$ denotes the set of directed connections between these registers. The *architecture-specific* CNOT *mapping problem* is stated as follows: Let $s, t \in R, s \neq t$ be two arbitrary registers in G^n. Find a mapping $M_{G^n} : \Gamma^n \to \Gamma^n$ s.t. M_{G^n} is a valid mapping for circuit $(\{\text{CNOT}(s,t)\})$. Specifically, we are interested in finding an efficient M_{G^n} that minimizes the complexity of $M_{G^n}(\{\text{CNOT}(s,t)\}))$ for a given G^n and (s,t) pair. In order to achieve this, we wish to develop mappings that minimize either the gate count or the depth of the resulting circuit.

We begin with a simpler version of the problem by imposing limitations on G^n. A *uni-directional line graph architecture* $G^n_{\Rightarrow}$ is an n-qubit architecture where all qubits are connected by a uni-directional path. To be concrete, let $G^n_{\Rightarrow} = (R, E)$ s.t. $R = (r_1, r_2, \ldots, r_n)$. Then $E = \{(r_1, r_2), (r_2, r_3), \ldots, (r_{n-1}, r_n)\}$. Similarly, a *bi-directional line graph architecture* $G^n_{\Leftrightarrow}$ is an n-qubit architecture where all qubits are connected by a bi-directional path. Let $G^n_{\Leftrightarrow} = (R, E)$ s.t. $R = (r_1, r_2, \ldots, r_n)$. Then $E = \{(r_1, r_2), (r_2, r_1), (r_2, r_3), (r_3, r_2), \ldots, (r_{n-1}, r_n), (r_n, r_{n-1})\}$. Starting with Section 4, we study efficient mappings of CNOT gates, specifically $(\{\text{CNOT}(r_1, r_n)\})$, on $G^n_{\Rightarrow}$ and $G^n_{\Leftrightarrow}$, before generalizing the results to arbitrary G^n in Section 7.

4 Lower Bounds

Before introducing specific algorithms about CNOT mapping on line graph architectures, we first propose and prove lower bounds on both gate count and circuit depth for mappings on $G^n_{\Rightarrow}$ and $G^n_{\Leftrightarrow}$. The bounds serve as a criterion for measuring the performance of the algorithms, and some intuition on finding efficient mappings may be won from the process of proving them.

In this section, we prove the lower bounds for a special case, in that all qubits are chosen from the set $\{|0\rangle, |1\rangle\}$. Note that lower bounds for a special case suffice as lower bounds for the general case. For ease of notation, we adopt a bitvector representation of quantum registers. Given an n-qubit architecture $G^n = (R, E)$ where $R = (r_1, r_2, \ldots, r_n)$, circuit C^n on n qubits that only consists of CNOT gates, initial assignments $\{r_1 \leftarrow |q_1\rangle, r_2 \leftarrow |q_2\rangle, \ldots, r_n \leftarrow |q_n\rangle\}$ such that $|q_1\rangle, |q_2\rangle, \ldots, |q_n\rangle \in \{|0\rangle, |1\rangle\}$, we use an n-bit vector for each register to denote the qubits that are "active" at this register. For each register r_i, we denote the bitvector of r_i by $v(r_i)$, and for the j-th bit of $v(r_i)$ we write $v_j(r_i)$. Initially, we zero the bitvector for each register. Then $\forall i \in [1, n]$, we set of the $v_(r_i)$, that is, bit i of the bitvector of register r_i to `1`. Now for each $\text{CNOT}(r_j, r_k)$ on pair (r_j, r_k), we set $v(r_k)$ to $v(r_k) \oplus v(r_j)$, where the operator $(\cdot) \oplus (\cdot)$ denotes the classical XOR operation. Recall that the classical XOR operation is commutative and associative, and $\forall x, y \in \mathbb{Z}_2$, we have the identity

$(x \oplus y) \oplus y = x$.

Under this notation, we note that the starting bitvector assignments are $v(r_1) = [\texttt{100}\ldots\texttt{000}]$, $v(r_2) = [\texttt{010}\ldots\texttt{000}]$, ..., $v(r_n) = [\texttt{000}\ldots\texttt{001}]$. Given $\text{CNOT}(r_1, r_n)$, the final assignments for $\{v(r_1), v(r_2), \ldots, v(r_{n-1})\}$ are the same as their starting assignments, while $v(r_n)$ should be modified to $[\texttt{100}\ldots\texttt{001}]$. This result should be exactly reproduced by any valid mapping of $\text{CNOT}(r_1, r_n)$ on an arbitrary G^n.

4.1 Uni-Directional Line Graph

4.1.1 Gate Count

We propose a $4n + O(1)$ lower bound on gate count for any mapping of a CNOT gate on $G^n_{\Rightarrow}$. Note that the technique described in Appendix A.3 is not considered here.

Theorem 1. *Given an n-qubit uni-directional line graph architecture $G^n_{\Rightarrow} = (R, E)$. For any mapping $M_{G^n_{\Rightarrow}}$ that is a valid mapping for $(\{\text{CNOT}(s,t)\})$ where $s, t \in R$, and that $M_{G^n_{\Rightarrow}}((\{\text{CNOT}(s,t)\}))$ does not contain Hadamard gates, the gate count of $M_{G^n_{\Rightarrow}}((\{\text{CNOT}(s,t)\}))$ is lower-bounded by $4n + O(1)$.*

It is helpful to first prove the following lemmas:

Lemma 1. *$\forall i \in [1, n]$, $\forall j \in [i+1, n]$, $v_j(r_i) = \texttt{0}$ at any point in the mapped circuit.*

Proof. The starting configuration satisfies $\forall i \in [1, n]$, $\forall j \in [i+1, n]$, $v_j(r_i) = \texttt{0}$. Observe that the only admissible CNOT operations are of the forward direction between neighboring qubits (i.e. $\text{CNOT}(r_i, r_{i+1})$ for $i \in [1, n-1]$). In order to modify $v_j(r_i)$ for some $i \in [1, n]$ and $j \in [i+1, n]$, it is necessary that $v_j(r_k) = 1$ for some $k < i$. This property telescopes and resolves to the requirement that $v_j(r_1) = \texttt{1}$. Since r_1 is not modified throughout the circuit, the requirement is not satisfiable. Therefore, we may conclude that $\forall i \in [1, n]$, $\forall j \in [i+1, n]$, $v_j(r_i) = \texttt{0}$ at any point in the mapped circuit. □

Lemma 2. *$\forall i \in [1, n]$, $v_i(r_i) = \texttt{1}$ at any point in the mapped circuit.*

Proof. Observe that in the starting configuration, $\forall i \in [1, n]$, $v_i(r_i) = \texttt{1}$. In order to modify $v_i(r_i)$ for some $i \in [1, n]$, it is required that $v_i(r_j) = \texttt{1}$ for some $j < i$. We use the result of Lemma 1 to conclude. □

Lemma 3. *$\forall i \in [1, n-1]$, at some point in the mapped circuit, $v_1(r_{i+1}) \neq v_i(r_{i+1})$.*

Proof. By way of contradiction, we assume that $\exists i \in [1, n-1]$ s.t. at any point in the mapped circuit, $v(r_{i+1})$ only takes assignments that contain either $v_1(r_{i+1}) = \texttt{1}$ and $v_i(r_{i+1}) = \texttt{1}$ simultaneously or $v_1(r_{i+1}) = \texttt{0}$ and $v_i(r_{i+1}) = \texttt{0}$ simultaneously.

Observe that in the final configuration, $v_1(r_n) = \texttt{1}$. Hence the only possible values that may satisfy the assumption are $i \in [2, n-2]$. Let r_m be a register

that satisfies the assumption. Then due to the neighboring and directional constraints of the CNOT gates it is easy to see that since $v_1(r_{m+1}) = v_m(r_{m+1})$ at any point in the circuit, $\forall j \geq m+1$, $v_1(r_j) = v_m(r_j)$ holds, at any point in the circuit. Observe that this contradicts the final assignment of r_n. Hence the assumption is false.

Therefore, we may conclude that $\forall i \in [1, n-1]$, at some point in the mapped circuit, $v_1(r_{i+1}) \neq v_i(r_{i+1})$. □

Lemma 4. *$\forall i \in [1, n-1]$, the number of instances of* $\text{CNOT}(r_i, r_{i+1})$ *in the mapped circuit is even.*

Proof. By Lemma 2, $\forall i \in [1, n]$, $v_i(r_i) = \texttt{1}$ at any point in the mapped circuit. Observe in the final configuration that $\forall i \in [1, n-1]$, $v_i(r_{i+1}) = \texttt{0}$. The conclusion then naturally follows from the properties of XOR. □

Proof of Theorem 1. In order for the set of registers to achieve its final configuration, $\forall i \in [2, n-2]$, $v(r_{i+1})$ satisfies the follows:

1. $v_1(r_{i+1}) = \texttt{1}$ at some point in the circuit, due to the constraints of the CNOT gates;
2. $v_i(r_{i+1}) = \texttt{1}$ at some point in the circuit, due to Lemma 2.

Observe that in the initial and final configurations, we have $v_1(r_{i+1}) = \texttt{0}$ and $v_i(r_{i+1}) = \texttt{0}$. Hence within the circuit, each of $v_1(r_{i+1})$ and $v_i(r_{i+1})$ must be reassigned twice. Observe that this requires at least two instances of $\text{CNOT}(r_i, r_{i+1})$. If indeed only two instances of $\text{CNOT}(r_i, r_{i+1})$ are present within the circuit, then the first instance sets both $v_1(r_{i+1})$ and $v_i(r_{i+1})$ to $\texttt{1}$, and the seconds instance sets both $v_1(r_{i+1})$ and $v_i(r_{i+1})$ to $\texttt{0}$. However, if this is the case, then the state of the two bits would violate Lemma 3. Hence by Lemma 4 we need at least 4 instances of $\text{CNOT}(r_i, r_{i+1})$. This gives us the $4n + O(1)$ lower bound on gate count. □

4.1.2 Circuit Depth

We propose a $2n + O(1)$ lower bound on circuit depth for any mapping of a CNOT gate on $G^n_{\Rightarrow}$. Note that the technique described in Appendix A.3 is not considered here.

Theorem 2. *Given an n-qubit uni-directional line graph architecture $G^n_{\Rightarrow} = (R, E)$. For any mapping $M_{G^n_{\Rightarrow}}$ that is a valid mapping for $(\{\text{CNOT}(s,t)\})$ where $s, t \in R$, and that $M_{G^n_{\Rightarrow}}(\{\text{CNOT}(s,t)\}))$ does not contain Hadamard gates, the circuit depth of $M_{G^n_{\Rightarrow}}(\{\text{CNOT}(s,t)\}))$ is lower-bounded by $2n+O(1)$.*

Proof. We first note that due to the neighboring constraint of the CNOT gates, it takes at least $n-1$ frames before $v_1(r_n)$ is set to $\texttt{1}$ for the first time. Now observe that the frame F_i where a $\text{CNOT}(r_{n-1}, r_n)$ sets $v_1(r_n)$ to $\texttt{1}$ requires $v_1(r_{n-1}) = \texttt{1}$ at that frame. In order to set $v_1(r_{n-1}) = \texttt{0}$ in compliance with the final configuration, it is then necessary for a $\text{CNOT}(r_{n-2}, r_{n-1})$ to occur

at some later frame F_j for some $j > i$. At F_j, it is further necessary that $v_1(r_{n-2}) = \mathtt{1}$. Therefore, we may easily see by induction that at least $n-2$ frames are required after F_i in order to set $v_1(r_k) = \mathtt{0}$ for all $k \in [2, n-1]$. Therefore, the total number of frames is at least $2n + O(1)$, hence is the lower bound on circuit depth. □

4.2 Bi-Directional Line Graph

4.2.1 Gate Count

We propose a $2n+O(1)$ lower bound on gate count for any mapping of a CNOT gate on $G^n_{\Leftrightarrow}$.

Theorem 3. *Given an n-qubit bi-directional line graph architecture $G^n_{\Leftrightarrow} = (R, E)$. For any mapping $M_{G^n_{\Leftrightarrow}}$ that is a valid mapping for $(\{\mathrm{CNOT}(s,t)\})$ where $s, t \in R$, the gate count of $M_{G^n_{\Leftrightarrow}}((\{\mathrm{CNOT}(s,t)\}))$ is lower-bounded by $2n + O(1)$.*

Proof. Due to the neighboring constraint of the CNOT gates, for each r_i s.t. $i \in [2, n-1]$, we have $v_1(r_i) = \mathtt{1}$ at some point in the mapped circuit. Since in the initial and final configurations $v_1(r_i) = \mathtt{0}$, we need at least two CNOT gates for each r_i, each setting $v_1(r_i)$ once. This gives the $2n + O(1)$ lower bound on gate count. □

4.2.2 Circuit Depth

We propose a $n+O(1)$ lower bound on circuit depth for any mapping of a CNOT gate on $G^n_{\Rightarrow}$.

Theorem 4. *Given an n-qubit bi-directional line graph architecture $G^n_{\Leftrightarrow} = (R, E)$. For any mapping $M_{G^n_{\Leftrightarrow}}$ that is a valid mapping for $(\{\mathrm{CNOT}(s,t)\})$ where $s, t \in R$, the circuit depth of $M_{G^n_{\Leftrightarrow}}((\{\mathrm{CNOT}(s,t)\}))$ is lower-bounded by $n + O(1)$.*

Proof. Due to the neighboring constraint of the CNOT gates, it takes at least $n-1$ frames before $v_1(r_n)$ is set to $\mathtt{1}$ for the first time. Hence we have an $n + O(1)$ lower bound on circuit depth. □

5 Heuristics

When designing an algorithm for performing CNOT mapping on a line graph architecture of n qubits, an important intuition is that $|q_1\rangle$ and $|q_n\rangle$ must be "brought together" to "meet" somewhere in the line graph. This intuition yields the $n + O(1)$ lower bound on circuit depth in Section 4.2. While doing so, however, one must also ensure that $|q_1\rangle$ and the set of intermediary qubits $\{|q_{[2,n-1]}\rangle\}$ are restored after the entire operation. This yields the $4n + O(1)$ lower bound on gate count in Section 4.1. An apparent method that satisfies both requirements is to preserve consistency at the local level. To be concrete,

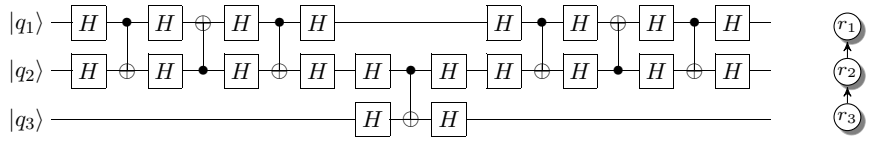

Figure 1: Uni-directional worst case on 3 qubits with the naïve SWAP heuristic

for each ordered pair $(|q_j\rangle, |q_k\rangle)$ of qubits of the non-terminal set $\{|q_{[1,n-1]}\rangle\}$, as stored in (r_i, r_{i+1}), we may implement a set of operations that allows $|q_j\rangle$ to be propagated to r_{i+1} while preserving $|q_k\rangle$. This can be done by utilizing the r_i position to store $|q_k\rangle$. We proceed to perform this set of operations for i from 1 to $n-2$. As $|q_1\rangle$ is now propagated to r_{n-1}, we may simply perform CNOT(r_{n-1}, r_n) to yield $|q_1\rangle \oplus |q_n\rangle$ on r_n. Afterwards, we repeat the same process in reverse order for (r_{n-1}, r_{n-2}) through (r_2, r_1), displacing $|q_1\rangle$ back to r_1 and all intermediary qubits to their original positions. An algorithm can be developed from this heuristic by implementing SWAP operations between each ordered pair of adjacent qubits. Since each SWAP gate consists of 3 CNOT gates, each operation has a circuit depth of 3 when a bi-directional connection is present between the two operands. When only a uni-directional connection is available, the circuit depth is 5 or 7, depending on the direction of connection (c.f. Appendix A.3). Altogether, This yields a circuit depth of $6n-11$ for a bi-directional line graph architecture, and a depth of between $10n-19$ and $14n-25$ for an arbitrary uni-directional line graph architecture. Figure 1 depicts a uni-directional worst case on 3 qubits, and Figure 2 depicts such a best case. The reader may verify that the output of both circuits is $\{r_1 \leftarrow |q_1\rangle, r_2 \leftarrow |q_2\rangle, r_3 \leftarrow |q_1\rangle \oplus |q_3\rangle\}$.

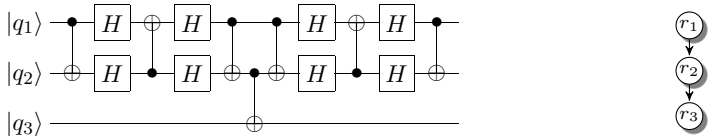

Figure 2: Uni-directional best case on 3 qubits with the naïve SWAP heuristic

In this example, consistency is preserved locally, in the sense that at any point in the circuit, each non-terminal register contains the information of only one unique qubit (considering SWAP as an atomic operation). In other words, at any point in the circuit, $\{r_{[1,n-1]}\}$ always contains a permutation of $\{|q_{[1,n-1]}\rangle\}$.

To optimize this heuristic, observe that for arbitrary qubits $|p\rangle$, $|q\rangle$ on single-qubit registers r, s, SWAP(r, s) and SWAP(s, r) both results in $\{r \leftarrow |q\rangle, s \leftarrow |p\rangle\}$. Hence we may re-orient the direction of SWAP operations to conform to the direction of connection between the pair of operand registers. This gives a circuit depth of 5 for each SWAP operation on any uni-directional line graph. Therefore, the worst-case circuit depth can be reduced to $10n-17$. The additive constant of 2 from the best-case $10n-19$ results from the extra Hadamard gates

surrounding CNOT(r_{n-1}, r_n).

A key observation that leads to further improvements is that preserving consistency at the local level is costly. In the naïve algorithm above, propagating each qubit $|q_i\rangle$ from register r_j to r_{j+1} costs 3 CNOT gates, while restoring the original information in r_{j+1} costs another 3. The number of CNOT gates required for each operation can be reduced if only partial consistency is needed. In this case, given ordered pair $(|q_j\rangle, |q_k\rangle)$ of qubits of the non-terminal set $\{|q_{[1,n-1]}\rangle\}$ stored on (r_i, r_{i+1}), we may allow r_{i+1} to store $f(|q_j\rangle, |q_k\rangle)$ for some Boolean function f as we propagate $|q_j\rangle$ to r_{i+1}. Alternatively, we may preemptively "back-up" $|q_k\rangle$ on r_i before the propagation, so that r_i yields $g(|q_j\rangle, |q_k\rangle)$ for some Boolean function g, and r_{i+1} in turn yields $f(g(|q_j\rangle, |q_k\rangle), |q_k\rangle)$ for some Boolean function f. Obviously, since local consistency is violated with these operations, additional operations should be in place to restore each r_i to its starting state after its information is propagated to the next register. The two heuristics are employed in designing the algorithms in Section 6.1 and Section 6.2 respectively.

So far, all heuristics covered involve propagating $|q_1\rangle$ from r_1 to r_n to facilitate $|q_1\rangle \oplus |q_n\rangle$ on r_n. However, it is also possible to propagate $|q_n\rangle$ from r_n backwards to r_i for some $i \in [1, n-1]$. By propagating $|q_1\rangle$ to the preceding neighbor r_{i-1}, one may produce $f(\{|q_1\rangle, |q_n\rangle\} \cup S)$ on r_i for some Boolean function f and some $S \subseteq \{|q_{[2,n-1]}\rangle\}$. Afterwards, we may then propagate the information on r_i to r_n to produce $|q_1\rangle \oplus |q_n\rangle$ on the latter. At the first glance, one may not discover how this approach fares better than the previous ones, as the steps of propagating $|q_n\rangle$ to r_i and then back to r_n seem redundant. The benefits of this method, however, depends on the fact that the tasks of propagating $|q_1\rangle$ to r_{i-1} and propagating $|q_n\rangle$ to r_i can be completed simultaneously, as the sets of registers required for either task are disjoint. Similarly, the tasks of restoring intermediary registers in disjoint ranges $[r_1, r_{i-1}]$ and $[r_i, r_n]$ can be completed in a parallel fashion. This heuristic facilitates the low-circuit-depth algorithm in Section 6.2.

6 Algorithms

In this section, we present two CNOT gate mapping algorithms on line graph architectures that are developed with the heuristics in Section 5.

The first of the two operates on a sequential basis, in that each frame of the circuit only contains one CNOT gate. This algorithm achieves a circuit depth of $4n - 8$ on a uni-directional line graph architecture, which constitutes a 2-approximation of the $2n + O(1)$ lower bound in Section 4.1. More notably, this algorithm meets the $4n + O(1)$ lower bound on gate count in Section 4.1 by an additive constant.

The second algorithm is parallel with two threads, in that at any frame of the circuit, there are either one or two CNOT gates. This algorithm achieves a gate count of $4n + O(1)$ and a circuit depth of $2n + O(1)$ on a bi-directional line graph architecture. These results constitute 2-approximations of the $2n + O(1)$

gate count and $n + O(1)$ circuit depth lower bounds in Section 4.2.

6.1 $(4n - 8)$-Deep Uni-Directional Mapping

We propose the $(4n - 8)$-deep uni-directional mapping algorithm. Note that for the sake of simplicity, we assume that the uni-directional architecture $G^n = (R, E)$ satisfies $E = \{(r_1, r_2), (r_2, r_3), \ldots, (r_{n-1}, r_n)\}$. In other words, all connections take a forward direction. To derive an algorithm for an arbitrary uni-directional architecture, one should utilize Hadamard gates in conjunction with reversed CNOT gates when necessary. The exact technique is described in Appendix A.3.

6.1.1 Description

We first give a formal description of the algorithm, after which we supplement the description with several examples.

Algorithm 1 $(4n - 8)$-deep uni-directional CNOT mapping

Input: Uni-directional line graph architecture $G^n_{\Rightarrow} = (R, E)$ where $E = \{(r_1, r_2), (r_2, r_3), \ldots, (r_{n-1}, r_n)\}$, set of qubits Q^n such that $|Q^n| = |R| = n$. Set of initial assignments $M^n_0 = \{r_1 \leftarrow |q_1\rangle, r_2 \leftarrow |q_2\rangle, \ldots, r_n \leftarrow |q_n\rangle\}$.
Output: List of frames $L^n = (F_1, F_2, \ldots, F_d)$, with each $F_i = \{g_{i1}, g_{i2}, \ldots, g_{ij}\}$ being a set of gates. Each g_{ik} is a CNOT$(\cdot, \cdot)$ legal with respect to G. The circuit represented by L^n achieves the set of final assignments $M^n = \{r_1 \leftarrow |q_1\rangle, r_2 \leftarrow |q_2\rangle, \ldots, r_{n-1} \leftarrow |q_{n-1}\rangle, r_n \leftarrow |q_1\rangle \oplus |q_n\rangle\}$, given M_0 as input.

$p \leftarrow 1, q \leftarrow 2n - 3, r \leftarrow 4n - 7$
$F_p \leftarrow \{\text{CNOT}(r_1, r_2)\}$
$F_q \leftarrow \{\text{CNOT}(r_1, r_2)\}$
for $i \in [1, n - 3]$ **do**
  $F_{p+i} \leftarrow \{\text{CNOT}(r_{i+1}, r_{i+2})\}$
  $F_{q-i} \leftarrow \{\text{CNOT}(r_{i+1}, r_{i+2})\}$
  $F_{q+i} \leftarrow \{\text{CNOT}(r_{i+1}, r_{i+2})\}$
  $F_{r-i} \leftarrow \{\text{CNOT}(r_{i+1}, r_{i+2})\}$
end for
$F_{p+n-2} \leftarrow \{\text{CNOT}(r_{n-1}, r_n)\}$
$F_{q+n-2} \leftarrow \{\text{CNOT}(r_{n-1}, r_n)\}$

6.1.2 Examples

When $n = 3$, the output of Algorithm 1 is as follows:

$$L^5 = (F_1, F_2, F_3, F_4) \tag{3}$$

with

$$F_1 = \{\text{CNOT}(r_1, r_2)\} \tag{4}$$
$$F_2 = \{\text{CNOT}(r_2, r_3)\} \tag{5}$$
$$F_3 = \{\text{CNOT}(r_1, r_2)\} \tag{6}$$
$$F_4 = \{\text{CNOT}(r_2, r_3)\}. \tag{7}$$

The output L^3 is shown in Figure 3.

Figure 3: Output of Algorithm 1 when $n = 3$.

When $n = 5$, the output of Algorithm 1 is as follows:

$$L^5 = (F_1, F_2, \ldots, F_{11}, F_{12}) \tag{8}$$

with

$$F_1 = F_7 = \{\text{CNOT}(r_1, r_2)\} \tag{9}$$
$$F_2 = F_6 = F_8 = F_{12} = \{\text{CNOT}(r_2, r_3)\} \tag{10}$$
$$F_3 = F_5 = F_9 = F_{11} = \{\text{CNOT}(r_3, r_4)\} \tag{11}$$
$$F_4 = F_{10} = \{\text{CNOT}(r_4, r_5)\}. \tag{12}$$

The output L^5 is shown in Figure 4.

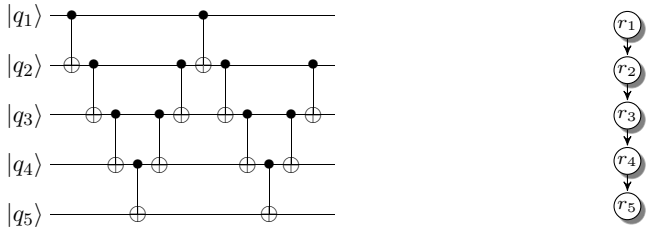

Figure 4: Output of Algorithm 1 when $n = 5$.

Observe that the algorithm produces a curious "zig-zag" arrangement of CNOT gates. Visually, the "zig-zag" forms a "W" shape, with the absence of a final $\text{CNOT}(r_1, r_2)$. Note that this pattern is preserved for any $n \geq 3$.

6.1.3 Proof of Correctness

We proceed to prove the correctness of Algorithm 1.

Theorem 5. *For all n, given $G^n_{\Rightarrow}$ and arbitrary Q^n, the output L^n produced by Algorithm 1 represents a circuit that takes M^n_0 as input and produces M^n as output.*

Remark. *It suffices to show that $\forall i \in [1, n-1]$, $r_i \leftarrow |q_i\rangle \in M^n$, and $r_n \leftarrow |q_1\rangle \oplus |q_n\rangle \in M^n$.*

Lemma 5. *$\forall i \in [1, n-1]$, $r_i \leftarrow |q_i\rangle \in M^n$, and $r_n \leftarrow |q_1\rangle \oplus |q_n\rangle \in M^n$ for Algorithm 1.*

Proof. First, Note that Algorithm 1 does not produce CNOT gates that modify r_1. Hence we have $r_1 \leftarrow |q_1\rangle \in M^n$. Next, observe that $\text{CNOT}(r_1, r_2) \in F_1$ and $\text{CNOT}(r_1, r_2) \in F_{2n-3}$ are the only two CNOT gates that modify r_2. Since r_1 remains unmodified, the two CNOT gates cancel. Hence $r_2 \leftarrow |q_2\rangle \in M^n$. Observe for $i \in [1, n-3]$, the CNOT gates that modify r_i are 4 $\text{CNOT}(r_{i+1}, r_{i+2})$ instances, respectively in $F_{p+i}, F_{q-i}, F_{q+i}$ and F_{r-i}. Since $\forall i$, $\{F_{[p+i+1,q-i-1]}\} \not\ni \{\text{CNOT}(r_i, r_{i+1})\}$, and $\forall i$, $\{F_{[q+i+1,r-i-1]}\} \not\ni \{\text{CNOT}(r_i, r_{i+1})\}$. r_i is not modified between F_{p+i} and F_{q-i}, and between F_{q+i} and F_{r-i}. Therefore, we may partition the 4 CNOT gates between r_{i+1} and r_{i+2} into 2 pairs, and the gates within either pair cancel. Hence we have $r_i \leftarrow |q_i\rangle \in M^n, \forall i \in [3, n-1]$ by shifting all indices by 2. □

Proof of Theorem 5. The correctness of Theorem 5 directly results from Lemma 5. □

6.1.4 Running Time

Observe that the algorithm performs constant work within the `for` loop for each i. Since the number of i is $O(n)$, the algorithm completes in $O(n)$.

6.1.5 Circuit Depth

The algorithm adds 4 frames for each $i \in [1, n-3]$ in the `for` loop. Additionally, the algorithm adds a constant of 4 frames. Therefore, the total amount of frames is $4n - 8$, hence is the circuit depth.

6.1.6 Gate Count

Observe that each frame produced by the algorithm only contains 1 CNOT gate. Therefore, the gate count is also $4n - 8$.

6.2 $(2n + O(1))$-Deep Bi-Directional Mapping

We propose the $(2n + O(1))$-deep bi-directional mapping algorithm. To achieve the $(2n + O(1))$ bound on circuit depth, we assume a bi-directional architecture $G^n = (R, E)$ that satisfies

$$E = \{(r_1, r_2), (r_2, r_3), \ldots, (r_{n-1}, r_n)\} \cup \{(r_2, r_1), (r_3, r_2), \ldots, (r_n, r_{n-1})\}. \quad (13)$$

In other words, bi-directional connections are present between each pair of adjacent registers. Again, Appendix A.3 describes a technique for generalizations on arbitrary uni-directional architectures.

Note that since the algorithm requires operations on 2 threads, the base case is given when $n = 4$.

6.2.1 Description

A formal description of the $(2n + O(1))$-deep bi-directional mapping algorithm is given as follows:

Algorithm 2 $(2n + O(1))$-deep bi-directional CNOT mapping

Input: Bi-directional line graph architecture $G^n_{\Leftrightarrow} = (R, E)$ where $n \geq 4$, $E = \{(r_1, r_2), (r_2, r_3), \ldots, (r_{n-1}, r_n)\} \cup \{(r_2, r_1), (r_3, r_2), \ldots, (r_n, r_{n-1})\}$, set of qubits Q^n such that $|Q^n| = |R| = n$. Set of initial assignments $M^n_0 = \{r_1 \leftarrow |q_1\rangle, r_2 \leftarrow |q_2\rangle, \ldots, r_n \leftarrow |q_n\rangle\}$.
Output: List of frames $L^n = (F_1, F_2, \ldots, F_d)$, with each $F_i = \{g_{i1}, g_{i2}, \ldots, g_{ij}\}$ being a set of gates. Each g_{ik} is a CNOT$(\cdot, \cdot)$ legal with respect to G. The circuit represented by L^n achieves the set of final assignments $M^n = \{r_1 \leftarrow |q_1\rangle, r_2 \leftarrow |q_2\rangle, \ldots, r_{n-1} \leftarrow |q_{n-1}\rangle, r_n \leftarrow |q_1\rangle \oplus |q_n\rangle\}$, given M_0 as input.

if $n \equiv 0 \mod 2$ **then**
  for $i \in [0, n/2 - 2]$ **do**
    $F_{2i+1} \leftarrow \{\text{CNOT}(r_{i+2}, r_{i+1}), \text{CNOT}(r_{n-i}, r_{n-i-1})\}$
    $F_{2i+2} \leftarrow \{\text{CNOT}(r_{i+1}, r_{i+2}), \text{CNOT}(r_{n-i-1}, r_{n-i})\}$
    $F_{2n-2i-3} \leftarrow \{\text{CNOT}(r_{i+2}, r_{i+1}), \text{CNOT}(r_{n-i}, r_{n-i-1})\}$
    $F_{2n-2i-4} \leftarrow \{\text{CNOT}(r_{i+1}, r_{i+2}), \text{CNOT}(r_{n-i-1}, r_{n-i})\}$
  end for
  $F_{n-1} \leftarrow \{\text{CNOT}(r_{n/2}, r_{n/2+1})\}$
else
  for $i \in [0, (n+1)/2 - 2]$ **do**
    $F_{2i+1} \leftarrow \{\text{CNOT}(r_{i+2}, r_{i+1}), \text{CNOT}(r_{n-i}, r_{n-i-1})\}$
    $F_{2i+2} \leftarrow \{\text{CNOT}(r_{i+1}, r_{i+2}), \text{CNOT}(r_{n-i-1}, r_{n-i})\}$
    $F_{2n-2i-2} \leftarrow \{\text{CNOT}(r_{i+2}, r_{i+1}), \text{CNOT}(r_{n-i}, r_{n-i-1})\}$
    $F_{2n-2i-3} \leftarrow \{\text{CNOT}(r_{i+1}, r_{i+2}), \text{CNOT}(r_{n-i-1}, r_{n-i})\}$
  end for
  $F_{n-2} \leftarrow \{\text{CNOT}(r_{(n-1)/2}, r_{(n+1)/2})\}$
  $F_{n-1} \leftarrow \{\text{CNOT}(r_{(n+1)/2}, r_{(n+3)/2})\}$
  $F_n \leftarrow \{\text{CNOT}(r_{(n-1)/2}, r_{(n+1)/2})\}$
  $F_{n+1} \leftarrow \{\text{CNOT}(r_{(n+1)/2}, r_{(n+3)/2})\}$
end if

6.2.2 Examples

We provide a few examples for both cases, namely $n \equiv 0 \mod 2$ and $n \equiv 1 \mod 2$.

When $n = 4$, the output of Algorithm 2 is as follows:

$$L^4 = (F_1, F_2, F_3, F_4, F_5) \tag{14}$$

with

$$F_1 = F_5 = \{\mathrm{CNOT}(r_2, r_1), \mathrm{CNOT}(r_4, r_3)\} \tag{15}$$
$$F_2 = F_4 = \{\mathrm{CNOT}(r_1, r_2), \mathrm{CNOT}(r_3, r_4)\} \tag{16}$$
$$F_3 = \{\mathrm{CNOT}(r_2, r_3)\}. \tag{17}$$

The output L^4 is shown in Figure 5.

Figure 5: Output of Algorithm 2 when $n = 4$.

When $n = 6$, the output of Algorithm 2 is as follows:

$$L^6 = (F_1, F_2, \ldots, F_8, F_9) \tag{18}$$

with

$$F_1 = F_9 = \{\mathrm{CNOT}(r_2, r_1), \mathrm{CNOT}(r_6, r_5)\} \tag{19}$$
$$F_2 = F_8 = \{\mathrm{CNOT}(r_1, r_2), \mathrm{CNOT}(r_5, r_6)\} \tag{20}$$
$$F_3 = F_7 = \{\mathrm{CNOT}(r_3, r_2), \mathrm{CNOT}(r_5, r_4)\} \tag{21}$$
$$F_4 = F_6 = \{\mathrm{CNOT}(r_2, r_3), \mathrm{CNOT}(r_4, r_5)\} \tag{22}$$
$$F_5 = \{\mathrm{CNOT}(r_3, r_4)\}. \tag{23}$$

The output L^6 is shown in Figure 6.

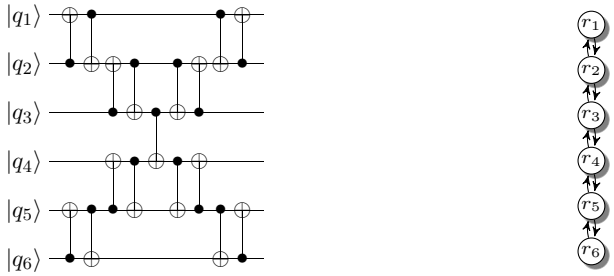

Figure 6: Output of Algorithm 2 when $n = 6$.

When $n = 5$, the output of Algorithm 2 is as follows:

$$L^5 = (F_1, F_2, \ldots, F_7, F_8) \tag{24}$$

with

$$F_1 = F_8 = \{\text{CNOT}(r_2, r_1), \text{CNOT}(r_5, r_4)\} \tag{25}$$
$$F_2 = F_7 = \{\text{CNOT}(r_1, r_2), \text{CNOT}(r_4, r_5)\} \tag{26}$$
$$F_3 = F_5 = \{\text{CNOT}(r_2, r_3\} \tag{27}$$
$$F_4 = F_6 = \{\text{CNOT}(r_3, r_4\}. \tag{28}$$

The output L^5 is shown in Figure 7.

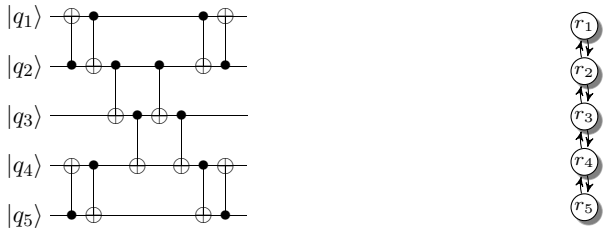

Figure 7: Output of Algorithm 2 when $n = 5$.

Visually, the arrangement of CNOT gates form an "X" shape. This pattern is also preserved for any $n \geq 4$.

6.2.3 Proof of Correctness

We proceed to prove the correctness of Algorithm 2.

Theorem 6. *For all n, given $G^n_{\Leftrightarrow}$ and arbitrary Q^n, the output L^n produced by Algorithm 2 represents a circuit that takes M^n_0 as input and produces M^n as output.*

Remark. *Similar to the proof of Theorem 5, it suffices to show that $\forall i \in [1, n-1]$, $r_i \leftarrow |q_i\rangle \in M^n$, and $r_n \leftarrow |q_1\rangle \oplus |q_n\rangle \in M^n$.*

Lemma 6. *$\forall i \in [1, n-1]$, $r_i \leftarrow |q_i\rangle \in M^n$, and $r_n \leftarrow |q_1\rangle \oplus |q_n\rangle \in M^n$ for Algorithm 2.*

Remark. *We first discuss the similarity between the two cases, namely $n \equiv 0 \mod 2$ and $n \equiv 1 \mod 2$. We then provide a proof that applies to both cases.*

Observe for the case of $n \equiv 1 \mod 2$, the only operations applied to $r_{(n+1)/2}$ are 4 CNOT operations located in frames F_{n-2} through F_{n+1}. Observe that within these frames, Algorithm 1 is performed on the ordered tuple $(r_{(n-1)/2}, r_{(n+1)/2}, r_{(n+3)/2})$. Therefore, these 4 CNOT gates constitute a CNOT gate from $r_{(n-1)/2}$ to $r_{(n+3)/2}$, with the content of $r_{(n+1)/2}$ unmodified after the sequence. Hence for any $n \geq 5$ such that $n \equiv 1 \mod 2$, proving Lemma 6 for n is equivalent to proving for $n - 1$.

Now it suffices to show that Lemma 6 holds $\forall n \geq 4$ s.t. $n \equiv 0 \mod 2$. We further separate this into two cases, in that we prove the following lemmas.

Lemma 7. *$\forall n \geq 4$ s.t. $n \equiv 0 \mod 2$, $\forall i \in [1, n/2]$, $r_i \leftarrow |q_i\rangle \in M^n$ for Algorithm 2.*

Lemma 8. *$\forall n \geq 4$ s.t. $n \equiv 0 \mod 2$, $\forall i \in [n/2+1, n-1]$, $r_i \leftarrow |q_i\rangle \in M^n$, and $r_n \leftarrow |q_1\rangle \oplus |q_n\rangle \in M^n$ for Algorithm 2.*

Proof of Lemma 7. We show $\forall i \in [1, n/2]$, $r_i \leftarrow |q_i\rangle \in M^n$ via induction. Observe as a base case that the only gates that modify $r_{n/2}$ are the two CNOT($r_{n/2-1}$, r_n) gates in frames F_{n-2} and F_n, and that between F_{n-2} and F_n no operation is conducted on $r_{n/2-1}$. Therefore, we have $r_{n/2} \leftarrow \left|q_{n/2}\right\rangle \in M^n$.

Next, observe that $\forall i \in [2, n/2-1]$, the gates that modify r_i are two CNOT(r_{i-1}, r_i) gates respectively in F_{2i-2} and F_{2n-2i}, in addition to the two CNOT(r_i, r_{i+1}) gates respectively in F_{2i-1} and $F_{2n-2i-1}$. Since F_{2i-1} and $F_{2n-2i-1}$ are the first and last frames that operate on r_{i+1}, we may conclude that if $r_{i+1} \leftarrow |q_{i+1}\rangle \in M^n$, then the contents of r_i before F_{2i-1} and after $F_{2n-2i-1}$ remain the same. Additionally, since no operation is conducted between F_{2i-2} and F_{2n-2i} on r_{i-1}, the two CNOT(r_{i-1}, r_i) gates in F_{2i-2} and F_{2n-2i} cancel. Therefore, we may derive the inductive step that $\forall i \in [2, n/2-1]$, if $r_{i+1} \leftarrow |q_{i+1}\rangle \in M^n$, then $r_i \leftarrow |q_i\rangle \in M^n$. By induction, we immediately have $\forall i \in [2, n/2]$, $r_i \leftarrow |q_i\rangle \in M^n$.

Finally, we handle the edge case where $i = 1$. Since $r_2 \leftarrow |q_2\rangle \in M^n$, and the only gates that modify r_1 are the two CNOT(r_2, r_1) gates in F_1 and F_{2n-3}, the two gates cancel and we have $r_1 \leftarrow |q_1\rangle \in M^n$.

This completes the proof of Lemma 7. □

Proof of Lemma 8. We first show that between F_{n-2} and F_{n-1}, $r_{n/2} \leftarrow |q_1\rangle$ via induction. As a base case, we know that $r_1 \leftarrow |q_1\rangle$ as initial assignment. Now assume that at some point in the circuit, we have $r_i \leftarrow |q_1\rangle$. After executing two adjacent frames that contain CNOT(r_{i+1}, r_i) and CNOT(r_i, r_{i+1}) respectively, we will obtain $r_{i+1} \leftarrow |q_1\rangle$. Hence by induction between F_{n-2} and F_{n-1} we have $r_{n/2} \leftarrow |q_1\rangle$.

Now we attempt to show that CNOT($r_{n/2}, r_{n/2+1}$) in F_{n-1}, together with all CNOT gates that operate on $\{r_{[n/2+1,n]}\}$, constitute CNOT($r_{n/2}, r_n$). We take the CNOT($r_{n/2}, r_{n/2+1}$) of interest as a base case. For the inductive step, assume for $i \in [n/2+2, n]$ that we have CNOT(r_i, r_{i-1}), CNOT(r_{i-1}, r_i), CNOT($r_{n/2}, r_{i-1}$), CNOT(r_{i-1}, r_i), and CNOT(r_i, r_{i-1}) in 5 adjacent frames. Given initial assignments $\{r_{n/2} \leftarrow |q_k\rangle,\ r_{i-1} \leftarrow |q_m\rangle,\ r_i \leftarrow |q_n\rangle\}$, it is easy to see that the end assignments after all 5 frames are executed is $\{r_{n/2} \leftarrow |q_k\rangle,\ r_{i-1} \leftarrow |q_m\rangle,\ r_i \leftarrow |q_k\rangle \oplus |q_m\rangle\}$. Therefore, these 5 adjacent frames constitute CNOT($r_{n/2}, r_i$). Since for each $i \in [n/2+2, n]$, the only gates between r_{i-1} and r_i are two CNOT(r_i, r_{i-1}) gates in $F_{2n-2i+1}$ and F_{2i-3}, and two CNOT(r_{i-1}, r_i) gates in $F_{2n-2i+2}$ and F_{2i-2}, we may induce that CNOT($r_{n/2}, r_{n/2+1}$) in F_{n-1}, together with all CNOT gates that operate on $\{r_{[n/2+1,n]}\}$, constitute CNOT($r_{n/2}, r_n$). This directly implies the initial assignments of $\{r_{[n/2+1,n-1]}\}$ are preserved after all frames are executed, and that

$r_n \leftarrow |q_n\rangle \oplus \left|q_{n/2}^{F_{n-1}}\right\rangle$, where $\left|q_{n/2}^{F_{n-1}}\right\rangle$ denotes the content of $r_{n/2}$ just before F_{n-1}. Since we have shown that between F_{n-2} and F_{n-1}, $r_{n/2} \leftarrow |q_1\rangle$, we may conclude that $r_n \leftarrow |q_1\rangle \oplus |q_n\rangle \in M^n$.

This completes the proof of Lemma 8. □

Proof of Lemma 6. The correctness of Lemma 6 is obtained by combining the results of Lemma 7 and Lemma 8. □

Proof of Theorem 6. The correctness of Theorem 6 directly results from Lemma 6. □

6.2.4 Running Time

Observe that for either case, the algorithm performs constant work within the `for` loop for each i. Since the number of i is $O(n)$, the algorithm completes in $O(n)$.

6.2.5 Circuit Depth

For the case of $n \equiv 0 \mod 2$, observe that the algorithm adds 4 frames for each $i \in [1, n/2-2]$, in addition to 1 extra frame that provides a $\mathrm{CNOT}(r_{n/2}, r_{n/2+1})$. Therefore, the total circuit depth is $2n-3$.

For the case of $n \equiv 1 \mod 2$, the algorithm adds 4 frames for each $i \in [1, (n+1)/2-2]$, in addition to 4 extra frames that provide the equivalent of $\mathrm{CNOT}(r_{(n-1)/2}, r_{n+3}/2)$. Therefore, the total circuit depth is $2n-2$.

Considering both cases, the circuit produced by Algorithm 2 is $(2n+O(1))$-deep.

6.2.6 Gate Count

For the case of $n \equiv 0 \mod 2$, the output produced by the algorithm contains $2n-8$ frames of two gates, and one frame of a single gate. Hence the total gate count is $4n-15$.

For the case of $n \equiv 1 \mod 2$, the output produced by the algorithm contains $2n-6$ frames of two gates, and 4 frames of a single gate. Hence the total gate count is $4n-8$.

Considering both cases, the circuit produced by Algorithm 2 has a total gate count of $4n+O(1)$.

7 Generalizations

In Section 6, the algorithms only produce mappings on line graph architectures. Specifically, recall that Algorithm 1 achieves its $4n-8$ circuit depth and gate count on a uni-directional line graph $G^n_{\Rightarrow}$ where all edges are forward, and Algorithm 2 achieves its $2n+O(1)$ circuit depth and $4n+O(1)$ gate count on a bi-directional line graph $G^n_{\Leftrightarrow}$.

Real-world quantum computers are often constructed in architectures that are very different from $G^n_{\Rightarrow}$ or $G^n_{\Leftrightarrow}$ [9]. Therefore, it is imperative to investigate how these algorithms could be applied in a general case.

Recall that in the general problem, given an arbitrary n-qubit architecture $G^n = (R, E)$, and let $s, t \in R$, $s \neq t$ be two arbitrary registers in G^n, we need to find an efficient mapping M_{G^n} for circuit $(\{\text{CNOT}(s,t)\})$. In order to leverage the results from Section 6, we partition the problem into the following steps:

1. In G^n, find a subgraph H^m with $m \leq n$ that is a line graph, with s and t being its end vertices.

2. Run Algorithm 1 or Algorithm 2 on H^m to produce mapped circuit L^m.

Specifically, we wish to ensure that for the algorithm of our choice, we find an H^m that minimizes the depth of the mapped circuit L^m. We do so by augmenting G^n with edge weights and finding the weighted shortest path between s and t. The weights we assign are nonnegative. Hence by applying Dijkstra's algorithm [2] with Fibonacci heap [4], we may achieve an efficient time complexity of $O(|E| + n \log n)$ for finding H^m, and an additional $O(m)$ for producing the mapped circuit L^m.

For each $(r_i, r_j) \in E$, the weight we assign to (r_i, r_j) corresponds to the number of frames it adds to L^n if it were to be included in H^m. We assign the same weight to (r_j, r_i), with an additional cost for CNOT gate reversal if $(r_j, r_i) \notin E$. We now propose the weight assignments for each algorithm.

7.1 Weights for the Uni-Directional Mapping Algorithm

Recall in Algorithm 1, $\forall i \in [2, n-2]$, there are 4 instances of $\text{CNOT}(i, i+1)$. Since reversing a CNOT gate requires 3 frames, we assign weight 1 to all forward edges of G^n, and weight 3 to all backward edges of G^n.

Formally, we augment G^n as follows: Create an empty edge set $E' = \varnothing$. $\forall (r_i, r_j) \in E$, assign weight 1 to (r_i, r_j), and if $(r_j, r_i) \notin E$, then add (r_j, r_i) to E' and assign weight 3 to it. Finally, merge E', along with the weight assignments of edges in it, into E.

Due to the 4 additional CNOT gates that connect r_1 with r_2, and r_{n-1} with r_n, these weight assignments guarantee optimality up to an additive constant.

7.2 Weights for the Bi-Directional Mapping Algorithm

In Algorithm 2, when $n \equiv 0 \mod 2$, $\forall i \in [1, n/2 - 1] \cup [n/2 + 1, n - 1]$, and when $n \equiv 1 \mod 2$, $\forall i \in [1, (n-3)/2] \cup [(n+3)/2, n-1]$, there are 2 instances of $\text{CNOT}(i, i+1)$ and 2 instances of $\text{CNOT}(i+1, i)$. Each pair of $(\text{CNOT}(i+1, i), \text{CNOT}(i, i+1))$ requires 4 frames when the connection between r_i and r_2 is uni-directional, and 2 when the connection is bi-directional. Therefore, we assign weight 2 to all edges of G^n, except when both an edge and its reverse exists–in which case, we assign weight 1 to both edges.

Formally, we augment G^n as follows: Create an empty edge set $E' = \varnothing$. $\forall (r_i, r_j) \in E$, if $(r_j, r_i) \in E$, assign weight 1 to both (r_i, r_j) and (r_j, r_i); otherwise, add (r_j, r_i) to E' and assign weight 2 to both (r_i, r_j) and (r_j, r_i). Finally, merge E', along with the weight assignments of edges in it, into E.

8 Conclusion

In this text, we introduced and provided a formal definition of the architecture-specific CNOT mapping problem. We studied specific cases of the problem on uni- and bi-directional line graph architectures, and provided lower bounds on gate count and circuit depth for either case. We introduced two algorithms that provide 2-approximations of the lower bounds on line graphs, and proposed efficient generalizations to arbitrary architectures. Additionally, we discussed general heuristics on approaching this problem, and how they lead to the discovery of the two algorithms.

The results we produced are applicable in designing efficient mechanisms that translate general quantum algorithms to sequences of instructions that are specific to a certain quantum computer architecture. Such mechanisms could enhance the workflow for designing, validating and deploying quantum algorithms, as well as the process of developing and maintaining quantum computers.

Future work on this topic includes proving the hardness of the general case of the CNOT mapping problem, investigating simultaneous mapping of multiple CNOT gates, developing optimizations for the arrangement of unary gates, and generalizing to full-circuit mapping on arbitrary architectures. We ultimately envision a future where quantum programs can be automatically "compiled" and "optimized" with respect to complex architectures, in the same fashion that classical computing has achieved today.

References

[1] Aleksandrowicz, G., Alexander, T., Barkoutsos, P., Bello, L., Ben-Haim, Y., Bucher, D., Cabrera-Hernádez, F. J., Carballo-Franquis, J., Chen, A., Chen, C.-F., Chow, J. M., Córcoles-Gonzales, A. D., Cross, A. J., Cross, A., Cruz-Benito, J., Culver, C., González, S. D. L. P., Torre, E. D. L., Ding, D., Dumitrescu, E., Duran, I., Eendebak, P., Everitt, M., Sertage, I. F., Frisch, A., Fuhrer, A., Gambetta, J., Gago, B. G., Gomez-Mosquera, J., Greenberg, D., Hamamura, I., Havlicek, V., Hellmers, J., Herok, Ł., Horii, H., Hu, S., Imamichi, T., Itoko, T., Javadi-Abhari, A., Kanazawa, N., Karazeev, A., Krsulich, K., Liu, P., Luh, Y., Maeng, Y., Marques, M., Martín-Fernández, F. J., McClure, D. T., McKay, D., Meesala, S., Mezzacapo, A., Moll, N., Rodríguez, D. M., Nannicini, G., Nation, P., Ollitrault, P., O'Riordan, L. J., Paik, H., Pérez, J., Phan, A., Pistoia, M., Prutyanov, V., Reuter, M., Rice, J., Davila, A. R., Rudy, R. H. P., Ryu, M., Sathaye, N., Schnabel, C., Schoute, E., Setia, K., Shi, Y., Silva, A., Siraichi, Y., Sivarajah, S., Smolin, J. A., Soeken, M., Takahashi, H., Tavernelli, I., Taylor, C., Taylour, P., Trabing, K., Treinish, M., Turner, W., Vogt-Lee, D., Vuillot, C., Wildstrom, J. A., Wilson, J., Winston, E., Wood, C., Wood, S., Wörner, S., Akhalwaya, I. Y., and Zoufal, C. Qiskit: An open-source framework for quantum computing, 2019.

[2] Dijkstra, E. W. A note on two problems in connexion with graphs. *Numer. Math. 1*, 1 (Dec. 1959), 269–271.

[3] Ekert, A., Hayden, P. M., and Inamori, H. Course 10: Basic Concepts in Quantum Computation. In *Coherent Atomic Matter Waves* (2001), R. Kaiser, C. Westbrook, and F. David, Eds., p. 661.

[4] Fredman, M. L., and Tarjan, R. E. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM 34*, 3 (July 1987), 596–615.

[5] IBM Q Team. IBM Q 16 Melbourne specification. `https://github.com/Qiskit/ibmq-device-information/tree/master/backends/melbourne/V1`, 2018.

[6] IBM Q Team. IBM Q 16 Rueschlikon specification. `https://github.com/Qiskit/ibmq-device-information/tree/master/backends/rueschlikon/V1`, 2018.

[7] IBM Q Team. IBM Q 5 Tenerife specification. `https://github.com/Qiskit/ibmq-device-information/tree/master/backends/tenerife/V1`, 2018.

[8] IBM Q Team. IBM Q 5 Yorktown specification. `https://github.com/Qiskit/ibmq-device-information/tree/master/backends/yorktown/V1`, 2019.

[9] IBM Q Team. IBM Q backend information. `https://github.com/Qiskit/ibmq-device-information`, 2019.

[10] Wille, R., Lye, A., and Drechsler, R. Exact reordering of circuit lines for nearest neighbor quantum architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 33*, 12 (Dec 2014), 1818–1831.

[11] Zulehner, A., Gasser, S., and Wille, R. Exact global reordering for nearest neighbor quantum circuits using A^*. pp. 185–201.

[12] Zulehner, A., Paler, A., and Wille, R. An Efficient Methodology for Mapping Quantum Circuits to the IBM QX Architectures. *arXiv e-prints* (Dec 2017), arXiv:1712.04722.

Acknowledgements

This project was completed under the supervision of Prof. Seth Lloyd of Department of Mechanical Engineering, MIT. The author would like to thank Prof. Lloyd for his essential guidance and support. Additionally, the author would like to thank Prof. Patrick Devlin of Yale University and Jiaqi Yu of Case Western Reserve University for valuable discussions that lead up to part of this project.

Appendix A Notations and Techniques

A.1 Disambiguation: Quantum Registers

Commonly, a quantum register is defined as a system comprising multiple qubits [3]. In this text, we put on a slightly different definition. We use the term "quantum register" to describe the position or the physical implement on a quantum computer that is capable of maintaining one qubit, and we reserve the term "qubit" for describing one quantum bit of information. Throughout this text, all mentions of a "quantum register" or "register" resolve to this definition. We use the lowercase $r_{(\cdot)}$ with subscript to denote a quantum register, and the bra-ket notation $\left|q_{(\cdot)}\right\rangle$ with subscript to denote a qubit. In line with this definition, when we discuss a quantum gate operation on a sequence of qubits $(|q_i\rangle, \ldots, |q_j\rangle)$, we first give an initial assignment of qubits to registers $\{r_k \leftarrow |q_i\rangle, \ldots, r_l \leftarrow |q_j\rangle\}$ and then denote the operation with respect to its operand registers (e.g. $\mathrm{CNOT}(r_k, r_l)$).

A.2 Notation Overloading with CNOT

For computational-basis qubits $|\alpha\rangle, |\beta\rangle \in \{|0\rangle, |1\rangle\}$, and $r_1 \leftarrow |\alpha\rangle$, $r_2 \leftarrow |\beta\rangle$ as initial assignments, the truth table of $\mathrm{CNOT}(r_1, r_2)$ is shown in Table 1.

Input		Output	
r_1	r_2	r_1	r_2
$\|0\rangle$	$\|0\rangle$	$\|0\rangle$	$\|0\rangle$
$\|0\rangle$	$\|1\rangle$	$\|0\rangle$	$\|1\rangle$
$\|1\rangle$	$\|0\rangle$	$\|1\rangle$	$\|1\rangle$
$\|1\rangle$	$\|1\rangle$	$\|1\rangle$	$\|0\rangle$

Table 1: Truth table of CNOT gate for inputs in $\{|0\rangle, |1\rangle\}$

Observe that the output on r_2 resembles the result of a classical XOR gate operating on inputs in the set $\{0, 1\}$. In light of this fact, for qubits $|\alpha\rangle$, $|\beta\rangle$ in the computational basis $\{|0\rangle, |1\rangle\}$, we write the result of $\mathrm{CNOT}(r_1, r_2)$ subject to assignment $\{r_1 \leftarrow |\alpha\rangle, r_2 \leftarrow |\beta\rangle\}$ as $\{r_1 \leftarrow |\alpha\rangle, r_2 \leftarrow |\alpha\rangle \oplus |\beta\rangle\}$, with the XOR operator $(\cdot) \oplus (\cdot)$.

A.3 CNOT Reversal

Given architecture $G^2 = (R, E)$, where $R = \{r_1, r_2\}$, $E = \{(r_1, r_2)\}$. A CNOT(r_2, r_1) on the computational basis $\{|0\rangle, |1\rangle\}$ can be implemented with a single CNOT(r_1, r_2) on the Hadamard transformed basis $\{|+\rangle, |-\rangle\}$. Specifically, we surrounded it with 4 Hadamard gates, as demonstrated in Figure 8.

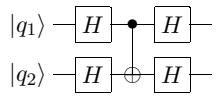

Figure 8: An implementation of CNOT(r_2, r_1) under (r_1, r_2) edge constraint

Note that a CNOT gate constructed in this way is 3-deep, as opposed to a normal 1-deep CNOT gate.