Improving the Fidelity of CNOT Circuits on NISQ Hardware

Dohun Kim[*†] dohunkim@snu.ac.kr
Minyoung Kim [*†] kimmy1110@snu.ac.kr
Sarah Meng Li[‡§¶‖**] sarah.li@uva.nl
Michele Mosca [§¶‖] michele.mosca@uwaterloo.ca

We introduce an improved CNOT synthesis algorithm that considers nearest-neighbour interactions and CNOT gate error rates in noisy intermediate-scale quantum (NISQ) hardware. Compared to IBM's Qiskit compiler, it improves the fidelity of a synthesized CNOT circuit by about 2 times on average (up to 9 times). It lowers the synthesized CNOT count by a factor of 13 on average (up to a factor of 162).

Our contribution is twofold. First, we define a Cost function by approximating the average gate fidelity F_{avg}. According to the simulation results, Cost fits the error probability of a noisy CNOT circuit, Prob $= 1 - F_{avg}$, much tighter than the commonly used cost functions. On IBM's fake Nairobi backend, it matches Prob to within 10^{-3}. On other backends, it fits Prob to within 10^{-1}. Cost accurately quantifies the dynamic error characteristics and shows remarkable scalability. Second, we propose a noise-aware CNOT routing algorithm, NAPermRowCol, by adapting the leading Steiner-tree-based connectivity-aware CNOT synthesis algorithms. A weighted edge is used to encode a CNOT gate error rate and Cost-instructed heuristics are applied to each reduction step. NAPermRowCol does not use ancillary qubits and is not restricted to certain initial qubit maps. Compared with algorithms that are noise-agnostic, it improves the fidelity of a synthesized CNOT circuit across varied NISQ hardware. Depending on the benchmark circuit and the IBM backend selected, it lowers the synthesized CNOT count up to 56.95% compared to ROWCOL and up to 21.62% compared to PermRowCol. It reduces the synthesis Cost up to 25.71% compared to ROWCOL and up to 9.12% compared to PermRowCol. Our method can be extended to route a more general quantum circuit, giving a powerful new tool for compiling on NISQ devices.

Contents

[*]Department of Physics and Astronomy, Seoul National University, Seoul, South Korea.
[†]Institute of Applied Physics, Seoul National University, Seoul, South Korea.
[‡]Informatics Institute, University of Amsterdam, Amsterdam, Netherlands.
[§]Department of Combinatorics & Optimization, University of Waterloo, Waterloo, Canada.
[¶]Institute for Quantum Computing , University of Waterloo, Waterloo, Canada.
[‖]Perimeter Institute for Theoretical Physics, Waterloo, Canada.
[**]Presenter

1 Introduction

The current phase of quantum technology is called "Noisy Intermediate-Scale Quantum" (NISQ) [56]. It is defined by the number of physical qubits (i.e., approximately between 50 and a few hundred) and their "noisy" nature (e.g., erroneous gate operation). Recently, the focus has shifted from merely increasing the number of qubits to enhancing their quality and error-correction capabilities [1, 11, 13, 53, 60, 61, 67, 72]. This marks a transition toward more reliable and practical quantum computing, namely, the "Logical Intermediate-Scale Quantum" (LISQ) [63].

Despite these pivotal changes, the current and near-future landscape is still dominated by NISQ devices [4, 5, 7, 38, 54, 55]. They allow researchers to explore the potential of quantum computers and carry out various tasks such as quantum simulation [8], combinatorial optimization [24], cryptography [31], and quantum chemistry [10]. Therefore, it is necessary and important to improve compilation methods tailored to NISQ devices and minimize the resource overhead [15, 30, 36].

Quantum circuit routing is the problem of mapping a logical circuit to NISQ hardware. The connectivity of NISQ architecture restricts a two-qubit gate to adjacent qubits, while various noise sources impact the reliability of quantum computations. An optimal solution to quantum circuit routing minimizes the resource overhead and maximizes the success probability of running a NISQ-executable circuit [17, 45, 46].

1.1 Problem of Interest

The main source of error in a NISQ device comes from entangling gates, whose error rates differ by a huge margin. These gates serve as a major component of many quantum programs. Although CNOT gates are not the native entangling operations in most NISQ hardware, they are commonly used in theoretical quantum circuits and algorithm designs. Since logical quantum circuits are transpiled to the device's native gate set and a CNOT gate is equivalent to a native entangling gate up to local operations, we consider CNOT gates for simplicity. For instance, IBM's quantum devices use the Echoed Cross-Resonance (ECR) gate as their entangling operation. It is equivalent to a CNOT gate up to single-qubit pre-rotations [20, 69]. Since CNOT error rate is about an order of magnitude higher than the single-qubit gate error rate [19, 46], the errors introduced by pre-rotations are relatively small. As a result, the CNOT error rate is about the same magnitude as that of an ECR gate.

Noise-aware CNOT circuit routing is a subproblem of quantum circuit routing where the logical circuits are composed of only CNOT gates. It accounts for the hardware connectivity and CNOT error rate to successfully route a CNOT circuit in a scalable manner [16, 74]. Although it is not immediately applicable for mapping a quantum program to NISQ devices, we can always decompose a quantum circuit into a layer of CNOT gates, followed by single-qubit gates, and then another layer of CNOT gates and so on [25]. Hence, in this paper, we focus on routing a noisy CNOT circuit, with the goal of reducing the resource overhead and improving the execution success probability.

CNOT circuits have well-behaved mathematical structures. The output parity terms of an n-qubit CNOT circuit correspond to an $n \times n$ parity matrix over $\mathbb{F}_2$, a non-singular binary square matrix. By using Gaussian elimination, we can decompose the parity matrix into a sequence of row operations, each of which corresponds to a CNOT gate [3, 52]. After concatenating these gates, we obtain a circuit with the same semantics as the input CNOT circuit (Section 2.2). This process is also called **CNOT circuit synthesis**.

Connectivity-aware CNOT circuit synthesis (a.k.a., CNOT circuit routing) takes a logical CNOT circuit and a uniform edge-weighted connectivity graph as inputs. It ignores the error distribution of different CNOT gates and returns a sequence of physically allowed CNOT operations. In literature, the connectivity constraint is also called the nearest-neighbour (NN) interaction. SWAP-based synthesis is one of the predominant methods to route a quantum circuit by relocating logical qubits in quantum registers [41, 66, 70, 73]. A major downside is that a SWAP gate equals three CNOT gates, so adding SWAP gates to quantum circuits results in an explosion of CNOT count.

Alternatively, Steiner-tree-based synthesis uses the parity matrix representation of a CNOT circuit and existing heuristics to optimize the synthesized CNOT count (Section 2.3). As a variant of a minimum-spanning tree, a Steiner tree finds the shortest path to connect a given set of vertices, corresponding to the shortest sequence of CNOT gates to route a subcircuit. Compared to simply inserting SWAP gates, reducing connectivity-aware CNOT circuit synthesis to a Steiner tree problem suppresses the CNOT explosion [25, 26, 27, 39, 48, 68].

For a quantum computer to be powerful, we should consider not only the idealized computation model, but also the imperfections and variations in the real system. Most of the leading Steiner-tree-based algorithms assume a uniform error distribution across the quantum system and use the synthesized CNOT count as their cost function. In practice, the error rate of each CNOT gate varies significantly depending on the coupled qubits' unique properties, their system positions, and the nature of the interactions they participate in. If certain qubits or connections have higher error rates, using them frequently might decrease the execution's accuracy, even when the overall gate count is reduced. For example, a routing path that minimizes CNOT count may choose more expensive edges (i.e., CNOT gates with higher error rates). It is unclear how closely the synthesized CNOT count aligns with the accumulated error in a noisy CNOT circuit.

1.2 Our Contributions

In this work, we focus on improving the fidelity of a NISQ-executable CNOT circuit. First, we approximate its average gate fidelity and propose a scalable Cost function to gauge its quality. In Section 3, we formally derive Cost by assuming CNOT gates are not parallelized and there is no noise on idle qubits. While these seem to be strong assumptions, we show that compared to the intuitively defined cost functions [16, 46, 74],

Cost fits the error probability of a noisy CNOT circuit tightly, for varied hardware topology and error distribution (Section 5.1). To the best of our knowledge, no one before us has investigated what is a good and scalable approximation of a CNOT circuit's reliability. Cost is the first attempt towards an efficient and accurate quantification of a noisy quantum circuit's reliability, rather than simply summing up the gate error rates or estimating the error probability disregarding the system size.

Next, we apply Cost to make a noise-aware adaptation of the algorithm PermRowCol [26], which is the leading Steiner-tree-based connectivity-aware CNOT circuit synthesis algorithm. In Section 4, we propose the algorithm NAPermRowCol to account for the connectivity and noise constraints. It not only returns a synthesized circuit with allowed CNOT operations and increased reliability, but also reduces the CNOT count by factoring out SWAP gates. Our technique can be summarized in two key points. First, it uses a noise-aware heuristic for pivot selection before each reduction step. Second, it prioritizes the cheapest way to route a noisy CNOT subcircuit. More precisely, it minimizes the cost to (1) propagate a parity 1 from one of all terminal nodes to a fixed Steiner node in each column reduction's first step, or (2) propagate the parity of a fixed Steiner node to one of all terminal nodes in each row reduction's first step.

Compared with GENNS [74], NAPermRowCol is not restricted to certain initial qubit maps. Compared with Qiskit [41], it uses no ancilla and is thus more resource-efficient. Compared to the leading CNOT routing algorithms, our benchmark results show that NAPermRowCol consistently improves the fidelity of a synthesized CNOT circuit across varied topologies (Section 5.2). Moreover, it reduces the synthesized CNOT count, shortening the circuit execution time.

1.3 Open Problems and Discussions

While noise-aware CNOT circuit routing is not immediately applicable to NISQ devices, our results pave the way to a fully noise-aware routing strategy. Here, we outline how NAPermRowCol could be extended to route a noisy quantum circuit over a universal gate set.

On one hand, we need an efficient cost function to quantify a noisy circuit's reliability. For noisy CNOT circuits, we use a generalized Pauli channel to model noise, assuming CNOT gates are not parallelized and idle qubits are noiseless. Although our empirical study shows that Cost approximates Prob closely, these simplifications make our cost function less general. A major room for improvement is to drop these assumptions.

For example, the generalized Pauli channels are a widely applicable model in quantum information, but they do not encompass all possible completely positive maps (e.g., complex multi-qubit interactions, non-Pauli types of noise and decoherence). Modelling a noisy quantum circuit with a more general channel representation is an important avenue for future work. Additionally, we should allow gate parallelization and account for errors resulting from the T1/T2 time. Moreover, we should consider single-qubit gate errors and readout errors arising from qubit state measurement. Depending on the quantum computing platform, we should also factor in error sources such as thermal relaxation and crosstalk.

In summary, the new cost function should accurately and efficiently quantify the dynamic error characteristics of a NISQ-executable circuit. Next, we can combine it with various circuit synthesis frameworks [25, 44, 47] to generalize NAPermRowCol and route an arbitrary quantum program on NISQ hardware. This includes designing noise-aware heuristics on a vertex-and-edge-weighted Steiner tree. We will then enhance the routed circuit's reliability by reducing the cost evaluation.

On the other hand, a routing algorithm's performance may decrease as the system scales with the qubit count and circuit complexity. For example, NAPermRowCol shows modest improvements when synthesizing a CNOT circuit over more than 16 qubits. This means we may need to refine the noise-aware heuristics to improve a routing algorithm's scalability.

1.4 Related Work

Substantial progress was made to understand noise within a quantum system [28, 29, 32, 50, 62, 64]. The problem of quantum circuit routing was introduced in [45]. Since then, numerous papers have appeared studying this problem [16, 35, 45, 46, 51, 58, 69, 74]. Most of them use intuitive cost functions without formal basis. As

cost function is a vital metric to improve the reliability of a NISQ-executable circuit, it is important to quantify a system's noise accurately and efficiently. In what follows, we describe in more detail those closely related to what we do.

[45] proposes a routing strategy tailored to Nuclear Magnetic Resonance (NMR) quantum computers. It uses circuit runtime as a cost function and inserts SWAP gates for quantum circuit routing. When testing its performance and scalability, the authors assume a linear architecture with a uniform interaction time between adjacent qubits. This is no longer a reasonable assumption for the leading NISQ devices. Moreover, NMR has largely been overshadowed by more practical quantum technologies, whose hardware topology is better suited for reliable and scalable quantum computing. For example, as of 2021, the topology of all active IBM quantum computers is based around the heavy-hex lattice. In each cell of the lattice, qubits are arranged in a hexagon, with an additional qubit on each edge [49]. Therefore, it is of pressing importance to develop noise-aware circuit routing strategies tailored to these state-of-the-art NISQ architectures.

[69] does not employ a cost function but instead suggests a noise-aware partition of a weighted connectivity graph as a preprocessor for IBM's Qiskit transpiler. According to user-defined error thresholds, it gets rid of disconnected vertices and graph components with high CNOT or readout error rates. Compared to Qiskit's inherent binary classification on whether a qubit is faulty or not, it takes advantage of the quantum processor's noise profile to make adaptive topology-pruning decisions.

[74] proposes the algorithm GENNS to route a CNOT circuit on IBM's NISQ devices. It enhances the routed circuit's reliability by accounting for the NN interactions and CNOT error rates. These restrictions are encoded in an edge-weighted connectivity graph. Among all pairs of connected qubits, GENNS uses the sum of edge weights as a cost function and applies the Floyd-Warshall algorithm [21] to find the shortest path. Since the cost function is proposed without a formal basis, GENNS may not return the most reliable routed circuit. Moreover, it is restricted to a feasible initial qubit map, or its reduction step terminates upon an invalid row operation. In the empirical study, GENNS is benchmarked with relatively short CNOT circuits (containing up to 256 gates) on 5- and 20-qubit backends. It is unclear how scalable and adaptive GENNS is for synthesizing a large CNOT circuit on varied IBM's backends.

[16, 46] propose comprehensive compiling strategies that are also customized for IBM's architecture. They encapsulate a noise-aware initial qubit mapping and a subsequent routing of the mapped circuit. [16] concentrates on routing a noisy CNOT circuit and proposes a Steiner-tree-based synthesis algorithm. It uses path fidelity (the product of CNOT success rates) as its cost function to instruct path selections in a Steiner tree. However, it is not clear how this cost function is related to the routed circuit reliability. [46] offers a collection of optimization- and heuristic-based methods to map an arbitrary quantum program to NISQ hardware. It accounts for CNOT and readout errors, as well as the connectivity and gate scheduling constraints. Its technique can be summarized in two key points. First, it reduces the problem of finding an optimal initial qubit mapping to a constrained optimization problem. Based on the linearized reliability score (i.e., the logarithm of the product of the CNOT gate and measurement success rates), it leverages the quantum analogue of the Satisfiability Modulo Theory (SMT) solver to find an optimal solution. Second, it proposes greedy heuristics that have comparable performance to the SMT-based methods, with improved scalability. Both approaches use SWAP gates for circuit routing and focus on an obsolete IBM grid topology in their empirical study.

2 Preliminaries

Here we review the core concepts for synthesizing a CNOT circuit on NISQ hardware. In Section 2.1, we introduce notions and conventions that will be used in this paper. In Section 2.2, we define the parity matrix representation of a CNOT circuit and use it to synthesize a noiseless CNOT circuit without any connectivity constraint. This is also known as the "CNOT circuit synthesis". In Section 2.3, we introduce the Steiner tree, a variant of the minimum spanning tree, and use it to synthesize a noiseless CNOT circuit based on a hardware topology. This is also known as the "connectivity-aware CNOT circuit synthesis". In Section 2.4, we define the average gate fidelity for a noisy quantum circuit and motivate noise-aware CNOT circuit routing on NISQ hardware.

2.1 Notations and Conventions

I denotes an identity operator (a.k.a, identity matrix), whose dimension can be inferred from the context. $\mathbb{C}$ denotes the set of complex numbers, $\mathbb{N}$ denotes the set of nonnegative integers, $\mathbb{N}^{\neq 0} = \mathbb{N} \setminus \{0\}$, and $\mathbb{Z}_q$ denotes the set of integers $\{0, 1, \ldots, q-1\}$. LHS (RHS) is short for the "lefthand (righthand) side" of an equation. $\mathsf{Tr}[A]$ denotes the trace of a matrix A. It is the sum of elements on the main diagonal of A. $A^\top$ denotes the transpose of matrix A. The Pauli matrices are 2×2 unitary operators acting on a single qubit. Let i be the imaginary unit, $i^4 = 1$.

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

By direct computation, $\mathsf{Tr}[I] = 2$, $\mathsf{Tr}[X] = \mathsf{Tr}[Y] = \mathsf{Tr}[Z] = 0$.

Definition 2.1. *For $n \in \mathbb{N}^{\neq 0}$, let $\mathcal{C}_n$ be the n-qubit Clifford group. $\mathcal{C}_n$ is generated by H, S, and CNOT gates through tensor product and composition. $\mathcal{C}_n = \langle H, S, CNOT \rangle$, where*

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \qquad S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}, \qquad CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{1}$$

Remark 2.1. $\mathcal{C}_1 = \langle H, S \rangle$.

CNOT gate is short for the "controlled-not gate". It is a quantum gate acting on two qubits. Its unitary matrix is shown in Equation (1). A **CNOT circuit** is a circuit composed of only CNOT gates. On top of the unitary matrix representation, there are different ways to represent a CNOT circuit. In this paper, they are used based on the problem that we are trying to solve. The parity matrix is often used to represent a logical CNOT circuit (Definition 2.3). The Kraus decomposition (Definition 2.16) and the superoperator representation (Lemma 2.7) are often used to represent a noisy CNOT circuit.

When the qubit count n increases, the parity matrix size grows polynomially with n, while the unitary matrix size grows exponentially. Therefore, the parity matrix representation is scalable and convenient for different instances of the CNOT routing problem. For example, it is used in CNOT circuit synthesis (Section 2.2), connectivity-aware CNOT circuit synthesis (Section 2.3), and noise-aware CNOT circuit routing (Section 4).

2.2 CNOT Circuit Synthesis

CNOT circuit synthesis takes a parity matrix as input and returns a CNOT circuit performing the desired operation [2, 52, 65]. It allows the coupling of any pair of qubits and aims to reduce the synthesized gate count. In what follows, we show that every CNOT circuit can be uniquely represented by a non-singular binary square matrix, namely, the parity matrix.

The parity matrix representation of a CNOT circuit Consider computational basis states $|c\rangle$ and $|t\rangle$, $c, t \in \mathbb{Z}_2$. Let $\oplus$ denote the bitwise XOR operation. $|c\rangle$ and $|t\rangle$ are called the **control and target qubit states**. When $|c\rangle = |0\rangle$, CNOT acts trivially on $|t\rangle$. Otherwise, CNOT acts as a NOT gate and flips $|t\rangle$. For convenience, the operation in Equation (2) is denoted as CNOT(c, t).

$$\text{CNOT}|c\rangle|t\rangle = |c\rangle|c \oplus t\rangle. \tag{2}$$

Definition 2.2. *Let $\boldsymbol{A}$ be a binary square matrix. In the matrices below, we use $'$ to distinguish between the indices for rows and columns. This convention will become useful in the CNOT circuit synthesis. When denoting a column operation $C(i', j')$ or column j', we drop $'$ for brevity.*

- *$R(c,t) \cdot \boldsymbol{A}$ denotes a **row operation** on $\boldsymbol{A}$, where row c is added to row t modulo 2 and row c remains unchanged. Let $[\![R(c,t)]\!]$ be the matrix representation of $R(c,t)$. Its diagonal components are equal to 1.*

Its off-diagonal components are equal to 0*, except for the entry on row t and column c.*

$$R(c,t)\cdot \boldsymbol{A} = [\![R(c,t)]\!]\boldsymbol{A}, \qquad [\![R(c,t)]\!] = \begin{array}{c} \\ \vdots \\ c \\ \vdots \\ t \\ \vdots \end{array} \begin{array}{c} \begin{array}{ccccc} \cdots & c' & \cdots & t' & \cdots \end{array} \\ \begin{pmatrix} & \vdots & & \vdots & \\ \cdots & 1 & \cdots & 0 & \cdots \\ & \vdots & & \vdots & \\ \cdots & 1 & \cdots & 1 & \cdots \\ & \vdots & & \vdots & \end{pmatrix} \end{array}.$$

- $\boldsymbol{A}\cdot C(c,t)$ *denotes a* ***column operation*** *on* $\boldsymbol{A}$*, where column c is added to column t modulo* 2 *and column c remains unchanged. Let* $[\![C(c,t)]\!]$ *be the matrix representation of* $C(c,t)$*. Its diagonal components are equal to* 1*. Its off-diagonal components are equal to* 0*, except for the entry on row c and column t.*

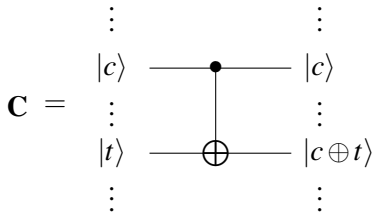
$$\boldsymbol{A}\cdot C(c,t) = \boldsymbol{A}[\![C(c,t)]\!], \qquad [\![C(c,t)]\!] = \begin{array}{c} \\ \vdots \\ c \\ \vdots \\ t \\ \vdots \end{array} \begin{array}{c} \begin{array}{ccccc} \cdots & c' & \cdots & t' & \cdots \end{array} \\ \begin{pmatrix} & \vdots & & \vdots & \\ \cdots & 1 & \cdots & 1 & \cdots \\ & \vdots & & \vdots & \\ \cdots & 0 & \cdots & 1 & \cdots \\ & \vdots & & \vdots & \end{pmatrix} \end{array}.$$

Definition 2.3. *A CNOT circuit over n qubits can be uniquely represented by an* $n \times n$ *binary square matrix, namely, the parity matrix. Let* $0 \le i, j \le n-1$*. The i-th row represents the i-the input qubit. The j-th column represents the parity term on the j-th output qubit [2, 65, 52].*

Example 2.1. *The parity matrix of an n-qubit empty circuit is an* $n \times n$ *identity matrix I.*

Example 2.2. *Figure 1a is a circuit consisting of one CNOT gate, CNOT(c, t). Figure 1b shows its parity matrix representation. CNOT(c, t) corresponds to performing a column operation C(c, t) on I, or performing a row operation R(t, c) on I.*

$$\boldsymbol{A} = I[\![C(c,t)]\!] = [\![R(t,c)]\!]I.$$



(a) **C** is an *n*-qubit CNOT circuit with one CNOT gate.

$$\mathbf{A} = \begin{array}{c} \\ \vdots \\ c \\ \vdots \\ t \\ \vdots \end{array} \begin{array}{c} \begin{array}{ccccc} \cdots & c' & \cdots & t' & \cdots \end{array} \\ \begin{pmatrix} & \vdots & & \vdots & \\ \cdots & 1 & \cdots & 1 & \cdots \\ & \vdots & & \vdots & \\ \cdots & 0 & \cdots & 1 & \cdots \\ & \vdots & & \vdots & \end{pmatrix} \end{array}$$

(b) **A** is the parity matrix of **C**.

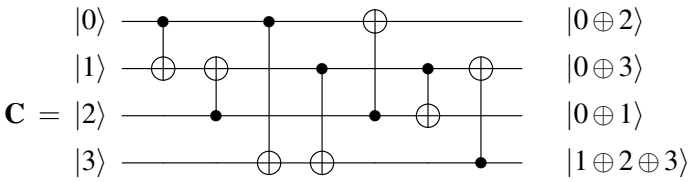
Figure 1: The action of a CNOT gate corresponds to a column (row) operation on I. This allows us to derive the parity matrix of a CNOT circuit.

Derive the parity matrix of a CNOT circuit To see the correspondence between a CNOT circuit and its parity matrix, consider a 4-qubit CNOT circuit in Figure 2a. Denote the initial state on each qubit wire by $|0\rangle$, $|1\rangle$, $|2\rangle$, and $|3\rangle$. On the righthand side of **C**, the output parity terms are $|0\oplus 2\rangle$, $|0\oplus 3\rangle$, $|0\oplus 1\rangle$, and $|1\oplus 2\oplus 3\rangle$. They are expressed by 4-dimensional binary vectors $\mathbf{b}_0$, $\mathbf{b}_1$, $\mathbf{b}_2$, and $\mathbf{b}_3$, where row i denotes the participation

of input qubit i in the parity term. More precisely, $\mathbf{b}_{ij} = 1$ indicates that input qubit i participates in the parity term j. $\mathbf{b}_{ij} = 0$ indicates otherwise. In Figure 2b, column j in $\mathbf{A}$ corresponds to $\mathbf{b}_j$.

$$\mathbf{b}_0 = \begin{bmatrix}1\\0\\1\\0\end{bmatrix}, \quad \mathbf{b}_1 = \begin{bmatrix}1\\0\\0\\1\end{bmatrix}, \quad \mathbf{b}_2 = \begin{bmatrix}1\\1\\0\\0\end{bmatrix}, \quad \mathbf{b}_3 = \begin{bmatrix}0\\1\\1\\1\end{bmatrix}.$$

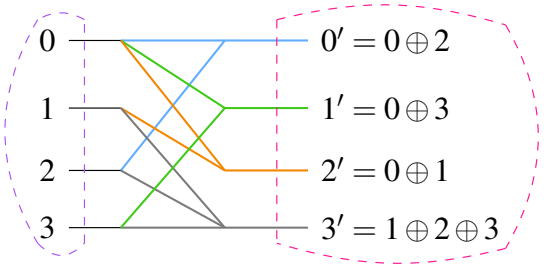
We can also express the entanglement in $\mathbf{C}$ as a bipartite graph. In Figures 2c and 2d, the inputs on wires 0, 1, 2, and 3 are entangled in the parity terms $0\oplus 2$, $0\oplus 3$, $0\oplus 1$, and $1\oplus 2\oplus 3$ on output wires $0'$, $1'$, $2'$, and $3'$ respectively. We use $'$ to denote an output wire of a quantum circuit. In the purple dashed box, $W_{in} = \{0,1,2,3\}$ denotes the input qubits. In the pink dashed box, $W_{out} = \{0',1',2',3'\}$ denotes the output qubits. W_{in} and W_{out} are two disjoint independent sets. The connectivity between them denotes the information propagation in $\mathbf{C}$. It is represented by the biadjacency matrix $\mathbf{A}$, which is the parity matrix of $\mathbf{C}$.



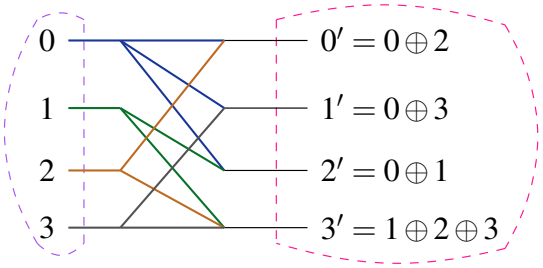
(a) $\mathbf{C}$ is a 4-qubit CNOT circuit.

$$\mathbf{A} = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0' & 1' & 2' & 3' \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix} \end{array}$$

(b) $\mathbf{A}$ is the parity matrix of $\mathbf{C}$.



(c) In the bipartite graph, edges with the same colour compose a parity term. Each colouring corresponds to a column in $\mathbf{A}$ (e.g., edges in light blue correspond to the first column and edges in light green correspond to the second column).



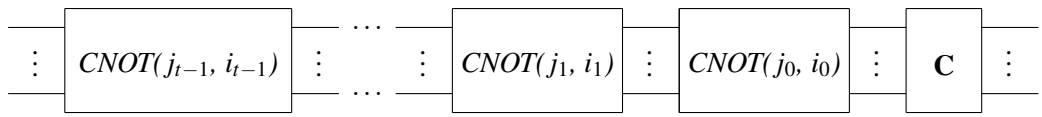
(d) In the bipartite graph, edges with the same colour denote the participation of an input qubit in all parity terms. Each colouring corresponds to a row in $\mathbf{A}$ (e.g., edges in dark blue correspond to the first row and edges in dark green correspond to the second row).

Figure 2: A 4-qubit CNOT circuit $\mathbf{C}$ is uniquely represented by a 4×4 parity matrix $\mathbf{A}$. Row i denotes the state on the input qubit wire i. Column j is the parity term $\mathbf{b}_j$ on the output qubit wire j'. The bipartite graph interpretation of $\mathbf{C}$ shows the information propagation in the CNOT circuit. Different ways of colouring edges help us interpret a row and column in $\mathbf{A}$.

Definition 2.4. *Let $\mathbf{C}$ be an n-qubit CNOT circuit with parity matrix $\mathbf{A}$. Right-concatenate $\mathbf{C}$ with a sequence of t CNOT gates, $t \in \mathbb{N}^{\neq 0}$. Let the resulting circuit be $\mathbf{C}_{syn}$. Without parallelizing CNOT gates, for $i_k, j_k \in \mathbb{Z}_n$, $k \in \mathbb{Z}_t$,*

$$\mathbf{C}_{syn} = \mathbf{C} \circ CNOT(j_0,i_0) \circ CNOT(j_1,i_1) \circ \ldots \circ CNOT(j_{t-1},i_{t-1}).$$

In the circuit diagram, it is visualized as follows.



Let $\mathbf{A}_{syn}$ be the parity matrix of $\mathbf{C}_{syn}$. $\mathbf{A}_{syn} = R(i_{t-1},j_{t-1}) \cdot \ldots \cdot R(i_1,j_1) \cdot R(i_0,j_0) \cdot \mathbf{A}$.

Exactly synthesize a CNOT circuit Next, we establish a one-to-one correspondence between a CNOT circuit and its parity matrix. Lemma 2.1 shows that every parity matrix has full rank. Based on this linear algebraic property, Lemma 2.2 proposes a way to exactly synthesize a CNOT circuit.

Lemma 2.1. *The parity matrix of an n-qubit CNOT circuit is an $n \times n$ binary matrix of full rank.*

Proof. By Definition 2.4, an n-qubit CNOT circuit corresponds to applying a sequence of row operations to I. Let the resulting matrix be $\mathbf{A}$. Since a square matrix is non-singular if and only if it is row-equivalent to an identity matrix, $\mathbf{A}$ is a binary square matrix of full rank. By Definitions 2.2 and 2.3, $\mathbf{A}$ is the parity matrix of $\mathbf{C}$. Hence, the parity matrix of a CNOT circuit is a binary square matrix of full rank. □

Let $\mathbf{A}$ be the parity matrix of an n-qubit CNOT circuit $\mathbf{C}$. By Lemma 2.1, $\mathbf{A}$ is a binary square matrix of full-rank. We can apply Gaussian elimination to find a sequence of row operations that sends $\mathbf{A}$ to the identity matrix I. Since each row operation corresponds to a CNOT gate, we can obtain a CNOT circuit $\mathbf{C}_{syn}$ by concatenating the corresponding CNOT operations designated by the Gaussian elimination. This process is called **CNOT circuit synthesis**. $\mathbf{C}_{syn}$ is the synthesized CNOT circuit of $\mathbf{A}$.

Lemma 2.2. *For $t \in \mathbb{N}^{\neq 0}$, $i_k, j_k \in \mathbb{Z}_n$, $k \in \mathbb{Z}_t$, let $R(i_0, j_0)$, $R(i_1, j_1)$, …, $R(i_{t-1}, j_{t-1})$ be a sequence of row operation on $\mathbf{A}$ such that*

$$R(i_{t-1}, j_{t-1}) \cdot \ldots \cdot R(i_1, j_1) \cdot R(i_0, j_0) \cdot \mathbf{A} = I. \tag{3}$$

Then $\mathbf{C}_{syn} = CNOT(j_{t-1}, i_{t-1}) \circ \ldots \circ CNOT(j_1, i_1) \circ CNOT(j_0, i_0)$ is a circuit representation of $\mathbf{A}$, so $\mathbf{C}$ and $\mathbf{C}_{syn}$ have the same semantics.

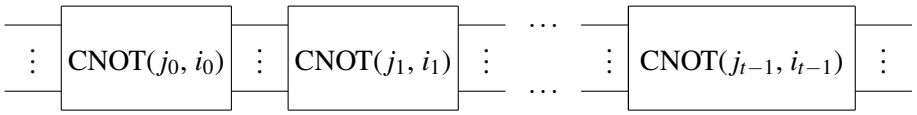
Proof. By Definition 2.4, Equation (3) is expressed as

$$\mathbf{C} \circ \big(\mathrm{CNOT}(j_0, i_0) \circ \mathrm{CNOT}(j_1, i_1) \circ \ldots \circ \mathrm{CNOT}(j_{t-1}, i_{t-1})\big) = I. \tag{4}$$

Right-multiplying Equation (4) by $\big(\mathrm{CNOT}(j_0, i_0) \circ \mathrm{CNOT}(j_1, i_1) \circ \ldots \circ \mathrm{CNOT}(j_{t-1}, i_{t-1})\big)^{-1}$ yields

$$\begin{aligned} \mathbf{C} &= \big(\mathrm{CNOT}(j_0, i_0) \circ \mathrm{CNOT}(j_1, i_1) \circ \ldots \circ \mathrm{CNOT}(j_{t-1}, i_{t-1})\big)^{-1} \\ &= \mathrm{CNOT}(j_{t-1}, i_{t-1}) \circ \ldots \circ \mathrm{CNOT}(j_1, i_1) \circ \mathrm{CNOT}(j_0, i_0) =: \mathbf{C}_{syn}. \end{aligned}$$

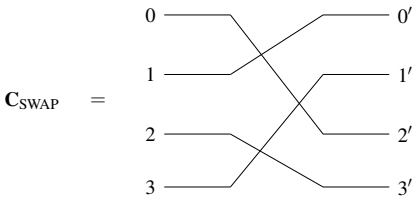
In the circuit diagram, $\mathbf{C}_{syn}$ is visualized as follows.

$\mathbf{C}_{syn} =$ 

□

Derive the parity matrix of a SWAP circuit A **permutation matrix** is a binary square matrix equivalent to an identity matrix up to row and column permutation. An $n \times n$ permutation matrix $\mathbf{P}$ represents a permutation of n elements. Let M be an $n \times n$ matrix. $\mathbf{P}$M permutes the rows of M, while M$\mathbf{P}$ permutes the columns of M.

A **SWAP circuit** is a circuit over SWAP gates. It can be obtained by relabelling the qubit wires. Since a SWAP gate is equal to three CNOT gates, a SWAP circuit is a CNOT circuit whose parity matrix coincides with a permutation matrix. In Figure 3, we use a 4-qubit SWAP circuit $\mathbf{C}_{\mathrm{SWAP}}$ to show that its parity matrix is exactly the permutation matrix for $\mathbf{C}_{\mathrm{SWAP}}$.



(a) Circuit $\mathbf{C}_{\text{SWAP}}$ is composed of a sequence of SWAP gates. As a result, the inputs on wires 0, 1, 2, and 3 are mapped to output wires 2′, 0′, 3′, and 1′ respectively. Equivalently, we can think of $\mathbf{C}_{\text{SWAP}}$ as a bipartite graph, with an input part $W_{in} = \{0,1,2,3\}$ and an output part $W_{out} = \{0',1',2',3'\}$. W_{in} and W_{out} are two disjoint independent sets. The connectivity between them shows the information propagation within $\mathbf{C}_{\text{SWAP}}$.

$$\mathbf{A} = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0' & 1' & 2' & 3' \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

(b) $\mathbf{A}$ is the parity matrix of $\mathbf{C}_{\text{SWAP}}$. It is the biadjacency matrix that describes the connectivity between W_{in} and W_{out} in the bipartite graph. It is known as a permutation matrix.

Figure 3: The parity matrix of an n-qubit SWAP circuit is an $n \times n$ permutation matrix.

Synthesize a CNOT circuit up to permutation Lastly, we generalize Lemma 2.2 to CNOT circuit synthesis up to permutation. This establishes the soundness of algorithm PermRowCol [26], based on which we develop the noise-aware CNOT routing algorithm NAPermRowCol in Section 4.

Lemma 2.3. *Let $\boldsymbol{A}$ be the parity matrix of an n-qubit CNOT circuit $\boldsymbol{C}$. For $t \in \mathbb{N}^{\neq 0}$ and $i_{t-1}, j_{t-1} \in \mathbb{Z}_n$, let $R(i_0,j_0)$, $R(i_1,j_1)$, …, $R(i_{t-1},j_{t-1})$ be a sequence of row operation on $\boldsymbol{A}$ such that*

$$R(i_{t-1},j_{t-1}) \cdot \ldots \cdot R(i_1,j_1) \cdot R(i_0,j_0) \cdot \boldsymbol{A} = \boldsymbol{P}, \tag{5}$$

$\boldsymbol{P}$ is the permutation matrix of a SWAP circuit $\boldsymbol{C}_{SWAP}$. Then

$$\boldsymbol{C}_{fully\ syn} = \boldsymbol{C}_{SWAP} \circ \boldsymbol{C}_{syn}, \qquad \boldsymbol{C}_{syn} = CNOT(j_{t-1},i_{t-1}) \circ \ldots \circ CNOT(j_1,i_1) \circ CNOT(j_0,i_0).$$

$\boldsymbol{C}_{fully\ syn}$ is a circuit representation of $\boldsymbol{A}$, so $\boldsymbol{C}$ and $\boldsymbol{C}_{fully\ syn}$ have the same semantics. In other words, $\boldsymbol{C}_{syn}$ is the synthesized CNOT circuit of $\boldsymbol{A}$ up to permutation.

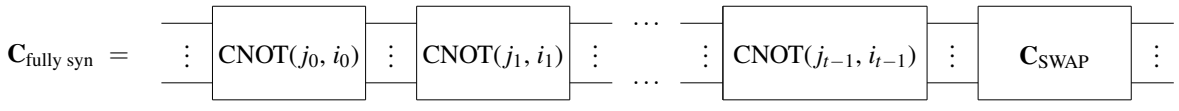
Proof. By Definition 2.4, Equation (5) is expressed as

$$\mathbf{C} \circ \big(\text{CNOT}(j_0,i_0) \circ \text{CNOT}(j_1,i_1) \circ \ldots \circ \text{CNOT}(j_{t-1},i_{t-1})\big) = \mathbf{C}_{\text{SWAP}}. \tag{6}$$

Right-multiplying Equation (6) by $\big(\text{CNOT}(j_0,i_0) \circ \text{CNOT}(j_1,i_1) \circ \ldots \circ \text{CNOT}(j_{t-1},i_{t-1})\big)^{-1}$ yields

$$\begin{aligned} \mathbf{C} &= \mathbf{C}_{\text{SWAP}} \circ \big(\text{CNOT}(j_0,i_0) \circ \text{CNOT}(j_1,i_1) \circ \ldots \circ \text{CNOT}(j_{t-1},i_{t-1})\big)^{-1} \\ &= \mathbf{C}_{\text{SWAP}} \circ \text{CNOT}(j_{t-1},i_{t-1}) \circ \ldots \circ \text{CNOT}(j_1,i_1) \circ \text{CNOT}(j_0,i_0) \\ &= \mathbf{C}_{\text{SWAP}} \circ \mathbf{C}_{syn} =: \mathbf{C}_{\text{fully syn}}. \end{aligned}$$

In the circuit diagram, $\mathbf{C}_{\text{fully syn}}$ is visualized as follows.



□

2.3 Connectivity-Aware CNOT Circuit Synthesis

Consider a NISQ hardware with at least n physical qubits. **Connectivity-aware CNOT synthesis** accounts for the hardware topology and synthesizes an n-qubit CNOT circuit based on an $n \times n$ parity matrix. In the meantime, it reduces the synthesized gate count. In Section 2.3.1, we introduce the Steiner tree, which is used to encode the connectivity constraint and efficiently synthesize a CNOT circuit. In Section 2.3.2, we reduce the connectivity-aware CNOT circuit synthesis to a Steiner tree problem. As a concrete example, we explain how algorithm PermRowCol [26] accounts for the connectivity constraint and synthesizes a CNOT circuit up to permutation.

2.3.1 Steiner Tree

Definition 2.5. *A **graph** G is defined by an ordered pair (V_G, E_G). V_G is a set of **vertices** and E_G is a set of **edges**. Each edge is defined as $e = (u, v)$, where $u, v \in V_G$. The **degree** of a vertex is the number of edges that are incident to this vertex.*

Definition 2.6. *Let $G = (V_G, E_G)$ be a graph.*

- *G is **edge-weighted** if it has a weight function $\omega_G : E_G \to \mathbb{R}$.*
- *G is **undirected** if none of its edges have orientations. For all $u, v \in V_G$, $(u, v) = (v, u)$.*
- *G is **connected** if every vertex in the graph is reachable from any other vertex by traversing a sequence of edges (i.e., a **path**). A **disconnected graph** is a graph that is not connected.*
- *$v \in V_G$ is a **cut vertex** if its removal disconnects G. It is a **non-cut vertex** if otherwise.*
- *G is **acyclic** if it has no cycle.*
- *G has a self-loop if $(u, u) \in E_G$ for some $u \in V_G$.*
- *G is a **tree** if it is undirected, connected, and acyclic.*

In this paper, we consider only the undirected edge-weighted connected graph and use $G = (V_G, E_G, \omega_G)$ to denote such graphs.

Definition 2.7. *A graph is **simple** if it is undirected with all edge weights equal to 1; it has at most one edge between two distinct vertices with no self-loops.*

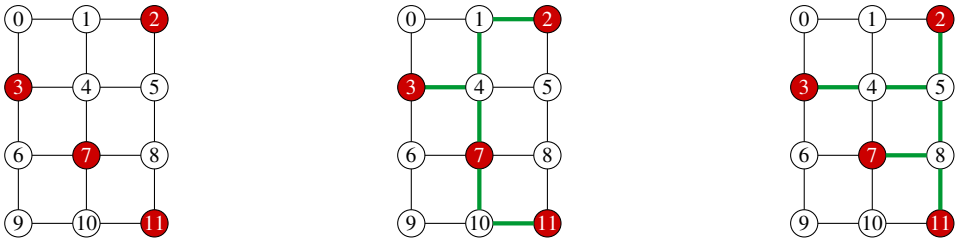
Definition 2.8. *A simple graph is **complete** if every pair of distinct vertices is connected by a unique edge. Otherwise, the graph is **incomplete**.*

Definition 2.9. *Let $G = (V_G, E_G, \omega_G)$.*

- *$H = (V_H, E_H, \omega_H)$ is a **subgraph** of G, denoted as $H \subseteq G$, if $V_H \subseteq V_G$, $E_H \subseteq E_G$, and $\omega_H(e) = \omega_G(e)$ for all $e \in E_H$.*
- *$T = (V_T, E_T, \omega_T)$ is a **minimum spanning tree** of G if $T \subseteq G$, $V_T = V_G$, with $\sum_{e \in E_T} \omega_T(e)$ minimal.*

Definition 2.10. *Let $G = (V_G, E_G, \omega_G)$, $S \subseteq V_G$. A **Steiner tree** $T = (V_T, E_T, \omega_T)$ is a subgraph of G such that $S \subseteq V_T$ with $\sum_{e \in E_T} \omega_T(e)$ minimal. S is called the **terminal**, the vertices in S are called the **terminal nodes**, and those in $V_T \setminus S$ are called the **Steiner nodes**. A solution to the **Steiner tree problem** Steiner(G, S) is a Steiner tree T of G with S as its leaves.*

A Steiner tree is a variation of a minimum-spanning tree. A Steiner tree problem involves finding a minimum-weight tree that spans a given set of vertices (i.e., the terminal nodes). Solutions to Steiner(G, S) are not unique. Consider an example in Figure 4, where G is a 12-vertex grid, $S = \{2, 3, 7, 11\}$, and $\omega_G : \mathbb{Z}_{12} \to \{1\}$. In Figure 4a, terminal nodes are coloured in red. Figures 4b and 4c show two distinct solutions to Steiner(G,S). The edges of the Steiner trees T_0 and T_1 are highlighted in green. The sets of Steiner nodes $V_{T_0} \setminus S$ and $V_{T_1} \setminus S$ can be read off from each graph.



(a) $S = \{2, 3, 7, 11\}$. (b) $V_{T_0} \setminus S = \{1, 4, 10\}$. (c) $V_{T_1} \setminus S = \{4, 5, 8\}$.

Figure 4: Solutions to a Steiner tree problem Steiner(G,S) are not unique. (a) G is a 12-vertex grid with $S = \{2, 3, 7, 11\}$. The terminal nodes are coloured in red. (b) T_0 is a solution to Steiner(G,S). The edges of the Steiner tree T_0 are coloured in green. Its Steiner nodes are the intermediary nodes in the Steiner tree, $V_{T_0} \setminus S = \{1, 4, 10\}$. (c) T_1 is an alternative solution to Steiner(G, S). Its set of Steiner nodes is $V_{T_1} \setminus S = \{4, 5, 8\}$.

Computing Steiner trees is NP-hard and the related decision problem is NP-complete [37]. There are different heuristic algorithms to compute approximate Steiner trees [14, 34, 59]. The choice of heuristics depends on their use case, as well as the trade-off between the problem's size and the algorithm's runtime.

2.3.2 Reduction to Steiner Tree Problem: PermRowCol

In superconducting quantum computers, qubits are arranged in a 2D grid. Each qubit can only interact with its nearest neighbours [1, 7, 23]. This is called the **NN interaction** and is modelled by a **connectivity graph**, $G = (V_G, E_G, \omega_G)$, $\omega_G : E_G \to \mathbb{R}$. Each vertex corresponds to a physical qubit, and each edge represents an entangling gate that can be performed on the qubits corresponding to its endpoints. Connectivity-aware CNOT circuit synthesis maps a logical CNOT circuit to NISQ hardware by accounting for its underlying topology, but assuming no noise in the system. That is, G is simple, $\omega_G : E_G \to \{1\}$.

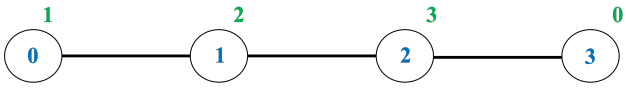
Let $\mathbf{C}$ be a logical CNOT circuit and $\mathbf{A}$ be its parity matrix. The synthesized circuit $\mathbf{C}_{\text{syn}}$ contains only CNOT gates acting on adjacent physical qubits in the NISQ hardware. Moreover, its CNOT count should be as few as possible. In [25, 26, 39, 48, 71], this problem is reduced to a Steiner tree problem. Here, we use the algorithm PermRowCol [26] to demonstrate the problem reduction. In Section 4, it is generalized to NAPermRowCol to account for noise in the system. In both cases, we assume an arbitrary initial qubit mapping, and it is illustrated below.

An arbitrary initial qubit mapping Let n be the number of logical qubits in $\mathbf{C}$ and $m = |V_G|$, $n \leq m$. Consider an initial qubit map where logical qubit i is mapped to quantum register j. Formally, this is defined by an injective function, $\Phi : \mathbb{Z}_n \to \mathbb{Z}_m$. $\Phi(\mathbb{Z}_n)$ corresponds to a connected subregion in the NISQ hardware. In what follows, we use vertices, physical qubits, and quantum registers interchangeably.

To illustrate the basics of PermRowCol, we use the 4-qubit CNOT circuit in Figure 6a with a linear topology in Figure 5b as an example. According to Figure 5a, the logical qubit i is mapped to the quantum register j by Φ, $\Phi : \mathbb{Z}_4 \to \mathbb{Z}_4$. This means we need to measure quantum register $\Phi(i)$ to return the state of logical qubit i. For clarity, we use a green label to denote a logical qubit and a blue label to denote a physical qubit.

Logical qubit i	0	1	2	3
Quantum register j	3	0	1	2

(a) Due to the initial qubit map, each logical qubit (in green) is mapped to a vertex (in blue).



(b) G is the linear graph. Each edge has a unit weight, which is omitted here. $\{0,3\}$ is the set of non-cut vertices.

Figure 5: Visualize the initial qubit map on a hardware topology.

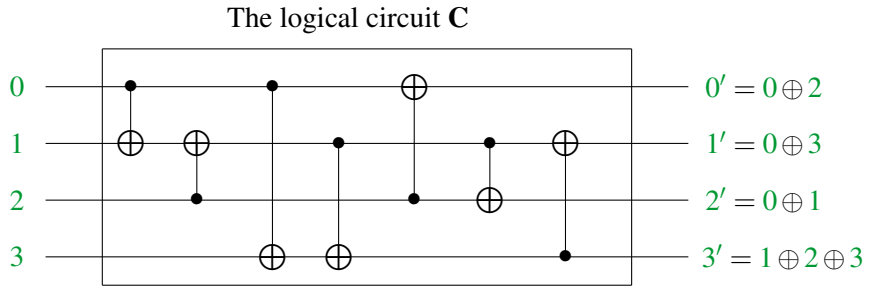
Remark 2.2. *The x-indexed row (a.k.a., row x) is a row whose index is x. It is not necessarily the x-th row in the matrix. For example, in Figure 6c, the 3-index row of $\mathbf{A}_0$ is the first row of $\mathbf{A}_0$. This convention also applies to the indexing of columns. On the physical layer, CNOT(i, j) denotes a CNOT gate acting on physical qubits labelled by i and j. Correspondingly, R(j, i) on $\mathbf{A}_0$ denotes a row operation of adding the j-indexed row to the i-indexed row, while keeping the j-indexed row unchanged.*

Figure 6b demonstrates the consequence of mapping each input qubit of $\mathbf{C}$ to a quantum register in Figure 5b. For brevity, we drop the ket notations in every circuit diagram and use $'$ to distinguish output wires from input wires.

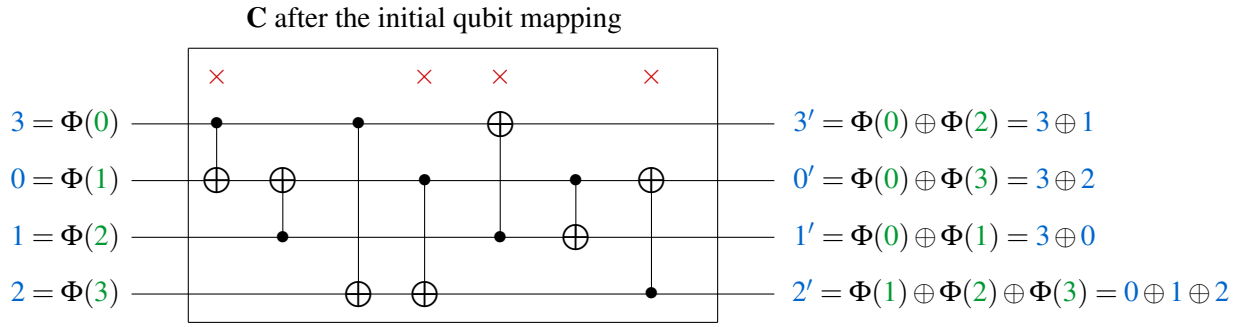
Definition 2.11. *Given a hardware topology G, CNOT(i, j) (or R(j, i) on $\mathbf{A}_0$) is **allowed** if quantum registers i and j are connected. It is **not allowed** if otherwise.*

In Figure 6b, some of the CNOT operations (annotated by $\times$) are not allowed because they do not act on adjacent physical qubits. For example, CNOT(3, 0) is not allowed because vertices 3 and 0 are not adjacent in Figure 5b. In light of this, we synthesize this circuit using PermRowCol, after which $\mathbf{A}_0$ is reduced to the permutation matrix $\mathbf{P}$,

$$\mathbf{P} = \begin{array}{c} \\ 3 \\ 0 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{cccc} 3' & 0' & 1' & 2' \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix} \end{array}.$$



(a) The logical circuit $\mathbf{C}$ is composed of a sequence of CNOT gates. The label of each wire is coloured green to indicate that it is a logical qubit. We use $'$ to distinguish output wires from input wires.



(b) After applying the initial qubit map Φ, the inputs of $\mathbf{C}$ are mapped to each quantum register in Figure 5b. On the physical layer, the labels of each wire are coloured blue. We use $'$ to distinguish output wires from input wires. Based on the linear topology, some of the CNOT operations are not allowed (annotated by $\times$) because they do not act on adjacent qubits.

$$\mathbf{A} = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0' & 1' & 2' & 3' \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix} \end{array} \xrightarrow{\Phi} \mathbf{A}_0 = \begin{array}{c} \\ 3 \\ 0 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{cccc} 3' & 0' & 1' & 2' \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix} \end{array}$$

(c) $\mathbf{A}$ and $\mathbf{A}_0$ are the parity matrices of $\mathbf{C}$ before and after the initial qubit mapping.

Figure 6: Φ maps the inputs of logical CNOT circuit $\mathbf{C}$ to quantum registers in NISQ hardware. Accordingly, $\mathbf{C}$'s parity matrix $\mathbf{A}$ is updated to $\mathbf{A}_0$. Connectivity-aware CNOT circuit synthesis is carried out on $\mathbf{A}_0$ with the hardware topology G in Figure 5b.

PermRowCol reduces a parity matrix to a permutation matrix In Section 2.2, Gaussian elimination reduces a parity matrix to an identity matrix. PermRowCol, however, reduces it to a permutation matrix. This means at each reduction step, we have more freedom to choose a pivot row and column. Here, we use the example in Figure 6 to provide detailed explanations of PermRowCol. Figure 7 demonstrates the synthesized circuit $\mathbf{C}_{\text{syn}}$ up to permutation of quantum registers.

In $\mathbf{C}_{\text{syn}}$, every CNOT operation is allowed. To recover the initial qubit mapping, $\mathbf{C}_{\text{syn}}$ is concatenated with a circuit over SWAP gates, $\mathbf{C}_{\text{SWAP}}$, whose parity matrix is $\mathbf{P}$. In the end, the fully synthesized circuit for $\mathbf{C}$ is expressed as $\mathbf{C}_{\text{SWAP}} \circ \mathbf{C}_{\text{syn}}$. We can read off an evolved state from a quantum register that is annotated on the right. For example, the evolved state $0 \oplus 2$ of the input wire 0 can be obtained by measuring quantum register

3 (denotes as $3''$). Without $\mathbf{C}_{\text{SWAP}}$, we can obtain this parity term by measuring quantum register 2 (denotes as $2'$). This is equivalent to relabeling quantum registers according to $\mathbf{P}$.

In a nutshell, PermRowCol reduces the CNOT count of a synthesized circuit by factoring out SWAP gates after synthesizing a parity matrix. It offloads the task of quantum computing onto classical processing and reduces the computation resources in NISQ compilation. As a result, $\mathbf{C}_{\text{syn}}$ contains fewer CNOT gates than the fully synthesized circuit $\mathbf{C}_{\text{SWAP}} \circ \mathbf{C}_{\text{syn}}$.

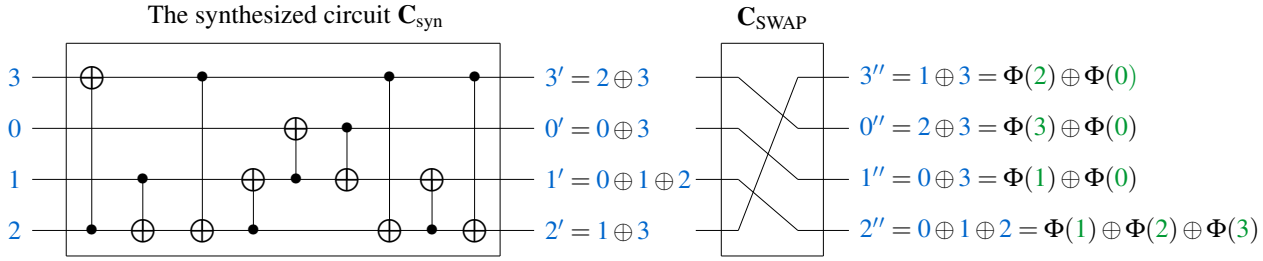


Figure 7: The fully synthesized circuit of $\mathbf{C}$ is expressed as $\mathbf{C}_{\text{SWAP}} \circ \mathbf{C}_{\text{syn}}$. The label of each wire is coloured blue (green) to indicate that it is a quantum register (logical qubit). $'$ and $''$ are used to distinguish output wires from input wires. For example, after $\mathbf{C}_{\text{syn}}$, quantum register 2 carries a parity term, $2' = 1 \oplus 3$. After $\mathbf{C}_{\text{SWAP}}$, this parity term is mapped to quantum register 3 , $3'' = 1 \oplus 3$. Equivalently, $\mathbf{C}_{\text{SWAP}}$ relabels quantum register 2 to 3. This means after $\mathbf{C}_{\text{SWAP}} \circ \mathbf{C}_{\text{syn}}$, we can measure quantum register 3 to obtain $1 \oplus 3 = \Phi(2) \oplus \Phi(0)$. In other words, the synthesized circuit successfully recovers the parity term in the output wire 0 of $\mathbf{C}$, up to permuting quantum registers.

The technicality of PermRowCol $\mathbf{A}_0$ is reduced to $\mathbf{P}$ through a sequence of reduction steps. Before each reduction, a pivot row and column (denoted as r and c) are selected based on the binary structure of $\mathbf{A}_0$ and the connectivity of G. In $\mathbf{A}_0$, suppose that row r corresponds to the i-th row and column c corresponds to the j-th column, $i, j \in \mathbb{Z}_n$. That is, $\Phi(i) = r$ and $\Phi(j) = c$. For $t \in \mathbb{Z}_n$, let e_t be the basis vector (i.e., the t-th column of I). A reduction step involves two actions by applying a sequence of row operations: a **column reduction** which transforms column c to e_i and a **row reduction** which transforms row r to $e_j^\top$. After a reduction step, the reduced row r and column c are removed from $\mathbf{A}_0$. Accordingly, vertex $\Phi(i)$ is removed from G. The algorithm terminates when there is one vertex left in G.

Procedure 2.1. *To select a pivot row, proceed as follows.*

1. *Among all rows in $\mathbf{A}_0$, find the set of rows whose indices correspond to the non-cut vertices in G. Let it be R_0.*
2. *Among all rows in R_0, find the set of rows with the minimum Hamming weight. Let it be R_1.*
3. *Return an arbitrary row in R_1. Let its index be r.*

Procedure 2.2. *Given the pivot row r, to select a pivot column, proceed as follows.*

1. *Among all columns in $\mathbf{A}_0$, find the set of columns with a non-zero entry at row r. Let it be C_0.*
2. *Among all columns in C_0, find the set of columns with the minimum Hamming weight. Let it be C_1.*
3. *Return an arbitrary column in C_1. Let its index be c.*

Column reduction Given a pivot column c, let S_0 be the set of its rows with a parity of 1. In the trivial case, column c has a hamming weight of 1 and $S_0 = \{r\}$. This means it is already the basis vector e_i. Otherwise, conduct Procedure 2.3 to perform column reduction.

Procedure 2.3. *Let $r \in S_0$ and $|S_0| > 1$. Build a Steiner tree T_0 of G where r is the root and S_0 is the terminal. Carry out two traversals in T_0, each of which returns a sequence of row operations.*

1. *Traverse T_0 from the leaves to the root. For each Steiner node v, add its child c to it. This corresponds to performing $R(c, v)$ on $\mathbf{A}_0$.*

2. *Traverse T_0 from the leaves to the root and add every parent p to its child c. This corresponds to performing R(p, c) on $\boldsymbol{A}_0$.*

Before the first traversal, the leaves of T_0 correspond to the rows (excluding row r) in column c that have a parity of 1, while the Steiner nodes correspond to the rows that have a parity of 0. After the first traversal, all Steiner nodes will carry a parity of 1. After the second traversal, the parity 1 from the root (i.e. row r) will be propagated to every other node in T_0. As a result, every row in column c will have a parity of 0 except for row r. Let ℓ_0 be a word composed of all row operations output from the two traversals in T_0. $t \in \mathbb{N}$,

$$\ell_0 = R(i_0, j_0)R(i_1, j_1)\dots R(i_{t-1}, j_{t-1}).$$

After column reduction, column c is reduced to e_i and $\mathbf{A}_0$ is updated as

$$\mathbf{A}_0 \leftarrow R(i_{t-1}, j_{t-1})\dots\cdot R(i_1, j_1)\cdot R(i_0, j_0)\cdot \mathbf{A}_0.$$

Row reduction In the trivial case, the pivot row r has a hamming weight of 1 and $S_1 = \{r\}$. This means it is already the basis vector $e_j^\top$. Otherwise, to reduce row r, we start by solving a system of linear equations: finding rows r_k in $\mathbf{A}_0$ such that $\bigoplus r_k = e_j^\top \oplus r$. Let S_1 be the set of these indices k including r.

Procedure 2.4. *Build a Steiner tree T_1 where r is the root and S_1 is the terminal. Carry out two traversals in T_1, each of which returns a sequence of row operations.*

1. *Traverse T_1 from the root to the leaves and add every Steiner node v to its parent p. This corresponds to performing R(v, p) on $\boldsymbol{A}_0$.*
2. *Traverse T_1 from the leaves to the root and add every child c to its parent p. This corresponds to performing R(c, p) on $\boldsymbol{A}_0$.*

Summing the parities of all rows in S_1 implies that all columns in row r carry a 0 except for column c.

$$\bigoplus_{k\in S_1} r_k = e_j^\top. \tag{7}$$

After the second traversal, the parity on each terminal node is propagated to the root and added together. Since the Steiner nodes are added twice modulo 2 throughout the two traversals, they do not participate in the parity sum of Equation (7). Let ℓ_1 be a word composed of all row operations output from the two traversals in T_1. $t' \in \mathbb{N}$,

$$\ell_1 = R(i_{0'}, j_{0'})R(i_{1'}, j_{1'})\dots R(i_{t'-1}, j_{t'-1}).$$

After row reduction, row r is reduced to $e_j^\top$ and $\mathbf{A}_0$ is updated as

$$\mathbf{A}_0 \leftarrow R(i_{t'-1}, j_{t'-1})\dots\cdot R(i_{1'}, j_{1'})\cdot R(i_{0'}, j_{0'})\cdot \mathbf{A}_0.$$

Update the parity matrix and the connectivity graph after a reduction step After a column reduction, the participation of all input registers except for quantum register r is removed from the output register c. After a row reduction, the participation of input register r is removed from all output registers except for the output register c. As a result, the input register r and output register c are no longer coupled with any other registers. This is illustrated in Figures 10 to 12.

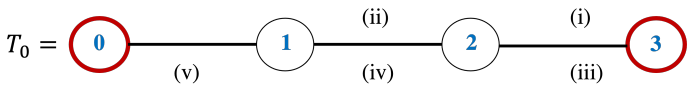
In the meantime, a parity term is eliminated to a single parity. Leveraging the reversibility of quantum gates and the relabelling of quantum registers, we can recover the evolved logical state in the designated quantum register. After that, row r and column c are removed from $\mathbf{A}_0$, and vertex r is removed from G. This means at each reduction step, our instance size is getting smaller. When PermRowCol terminates, we can recover the permutation matrix $\mathbf{P}$ by assembling the reduced row and column from each reduction step.

Demonstrate a reduction step To illustrate PermRowCol's technicality, we use it to synthesize the logical circuit in Figure 6a according to the hardware topology in Figure 5b. Since this is carried out on the physical layer, there is no ambiguity and we will no longer use coloured labels in the descriptions below.

Figure 8 demonstrates the column reduction in the first reduction step, where column 1 is eliminated. In Figure 8b, Steiner tree T_0 is constructed based on the pivot column, with vertex 0 as the root and vertex 3 as the leave. To reduce column 1, we carry out two traversals. The first bottom-up traversal returns two row operations R(3,2) and R(2,1). The second bottom-up traversal returns three row operations R(2,3), R(1,2), and R(0,1). Hence $\ell_0 = R(3,2)R(2,1)R(2,3)R(1,2)R(0,1)$. In Figure 8c, they are performed on $\mathbf{A}_0$ and we get the evolved parity matrix $\mathbf{A}_0^0$ after the column reduction.

$$\mathbf{A}_0 = \begin{array}{c} \\ 3 \\ 0 \\ 1 \\ 2 \\ \text{Candidate columns} \end{array} \begin{array}{c} \begin{array}{cccc} 3' & 0' & 1' & 2' \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix} \\ \begin{array}{cccc} & & 2 & 3 \end{array} \end{array} \begin{array}{c} \text{Candidate rows} \\ 3 \\ 2 \end{array}$$

(a) For the first reduction step, the pivot row and column are highlighted in red.



(b) Steiner tree T_0 is constructed for the column reduction. Vertex 0 is the root and vertex 3 is the leaf. $S = \{0,3\}$. Two traversals of T_0 return a sequence of row operations, (i) $\sim$ (v).

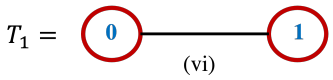
$$\mathbf{A}_0 \xrightarrow[R(3,2)]{\text{(i)}} \begin{array}{c} \\ 3 \\ 0 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{cccc} 3' & 0' & 1' & 2' \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix} \end{array} \xrightarrow[R(2,1)]{\text{(ii)}} \begin{array}{c} \\ 3 \\ 0 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{cccc} 3' & 0' & 1' & 2' \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \end{array} \xrightarrow[R(2,3)]{\text{(iii)}} \begin{array}{c} \\ 3 \\ 0 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{cccc} 3' & 0' & 1' & 2' \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \end{array} \xrightarrow[R(1,2)]{\text{(iv)}}$$

$$\begin{array}{c} \\ 3 \\ 0 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{cccc} 3' & 0' & 1' & 2' \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \end{array} \xrightarrow[R(0,1)]{\text{(v)}} \begin{array}{c} \\ 3 \\ 0 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{cccc} 3' & 0' & 1' & 2' \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix} \end{array} = \mathbf{A}_0^0.$$

(c) After traversing T_0, perform a sequence of row operations on $\mathbf{A}_0$. Column 1 is reduced to e_1 and $\mathbf{A}_0^0$ is the evolved parity matrix.

Figure 8: Illustrate the column reduction in the first reduction step.

After that, Figure 9 demonstrates the row reduction in the first reduction step, where row 0 is eliminated. In Figure 9a, Steiner tree T_1 is constructed based on the pivot row in $\mathbf{A}_0^0$, since $r \oplus r_1 = e_2^\top$. The tree traversals only return a row operation R(1,0). Hence $\ell_1 = R(1,0)$. In Figure 9b, it is performed on $\mathbf{A}_0^0$ and we get the evolved parity matrix $\mathbf{A}_0^1$ after the row reduction.



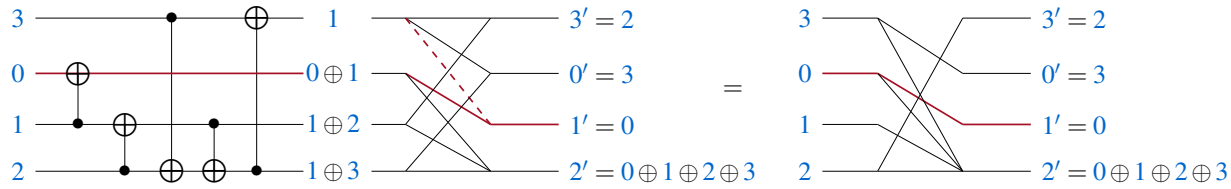
(a) Steiner tree T_1 is constructed for the row reduction. Vertex 0 is the root and vertex 1 is the leaf. $S = \{0,1\}$. Two traversals of T_1 return one row operation, (vi).

$$\mathbf{A}_0^0 \xrightarrow[R(1,0)]{\text{(vi)}} \begin{array}{c} \\ 3 \\ 0 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{cccc} 3' & 0' & 1' & 2' \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix} \end{array} = \mathbf{A}_0^1.$$

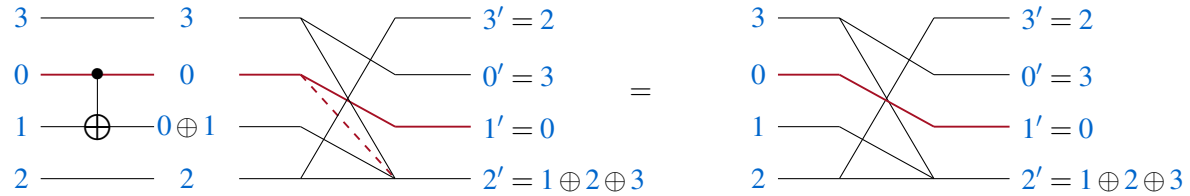
(b) After traversing T_1, perform a row operation on $\mathbf{A}_0^0$. Row 0 is reduced to $e_2^\top$ and $\mathbf{A}_0^1$ is the evolved parity matrix.

Figure 9: Illustrate the row reduction in the first reduction step.

Figure 10 articulates the intuition for a column and row reduction using the bipartite graph introduced in Section 2.2. Since the input register 0 and output register 1 are no longer coupled with any register, row 0 and column 1 are removed from $\mathbf{A}_0^1$. The parity matrix $\mathbf{A}_0$ is updated in Figure 11a. In the meantime, vertex 0 is removed from G and the connectivity graph is updated in Figure 11b. Figures 11 and 12 demonstrate details of the remaining reduction steps.



(a) In the column reduction, a sequence of row operations are performed on $\mathbf{A}_0$. They correspond to left-applying a sequence of CNOT gates to circuit $\mathbf{C}$ after the initial qubit mapping, whose information propagation is illustrated by the bipartite graph (LHS). The pivot row corresponds to the input register 0. The pivot column corresponds to the output register 1. Relevant qubit wires are highlighted in red. After the column reduction, the updated coupling of circuit $\mathbf{C}_0^0$ (RHS) is described by $\mathbf{A}_0^0$. Except for quantum register 0 (the red solid wire), the column reduction removes all other input registers (the red dashed wire) from output register 1.



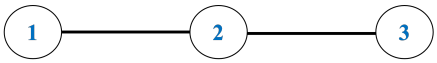
(b) In the row reduction, a row operation R(1,0) is performed on $\mathbf{A}_0^0$. It corresponds to left-applying CNOT(0,1) to circuit $\mathbf{C}_0^0$, whose information propagation is illustrated by the bipartite graph (LHS). After the row reduction, the updated parity matrix $\mathbf{A}_0^1$ corresponds to the updated circuit $\mathbf{C}_0^1$, whose information propagation is illustrated by the bipartite graph (RHS). The row reduction removes input register 0 from all other output registers (the red dashed wire), except output register 1(the red solid line).

Figure 10: The intuition behind the first reduction step.

Lastly, we comment on the heuristic algorithm that we choose to create a Steiner tree with a terminal S for a simple connected graph. This is equivalent to constructing a minimum spanning tree over S using Dijkstra's Shortest Path algorithm [33]. For each terminal node, its shortest path to the root is used to construct the minimum spanning tree. After that, the calculation of total edge weight considers the paths between the constructed tree and the terminals that have not yet been added to the spanning tree.

$$\mathbf{A}_0 = \begin{array}{c} \\ 3 \\ 1 \\ 2 \\ \text{Candidate columns} \end{array} \begin{array}{c} \begin{array}{ccc} 3' & 0' & 2' \end{array} \\ \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix} \\ \begin{array}{ccc} & & 3 \end{array} \end{array} \begin{array}{c} \text{Candidate rows} \\ 2 \\ 1 \\ \\ \end{array}$$

(a) For the second reduction step, the pivot row and column are highlighted in red.

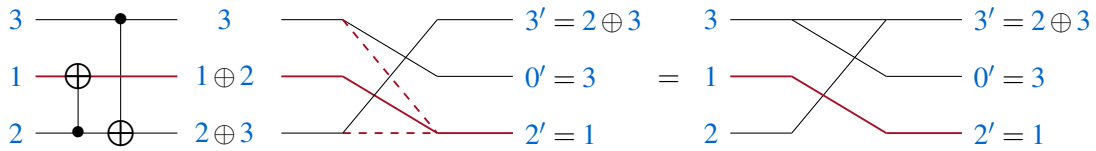


(b) G is the updated connectivity graph after removing vertex 0.

(c) Steiner tree T_0 is constructed for the column reduction. Vertex 1 is the root and vertex 3 is the leaf. $S = \{1,3\}$. Two traversals of T_0 return a sequence of row operations (annotated by (i) $\sim$ (ii)).

$$\mathbf{A}_0 \xrightarrow[R(2,3)]{\text{(i)}} \begin{array}{c} \\ 3 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{ccc} 3' & 0' & 2' \end{array} \\ \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix} \end{array} \xrightarrow[R(1,2)]{\text{(ii)}} \begin{array}{c} \\ 3 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{ccc} 3' & 0' & 2' \end{array} \\ \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \end{array} = \mathbf{A}_0^0 = \mathbf{A}_0^1.$$

(d) After traversing T_0, perform a sequence of row operations on $\mathbf{A}_0$. Column 2 is reduced to e_2 and $\mathbf{A}_0^0$ is the evolved parity matrix. Since row 1 is also reduced to $e_3^\top$, $\mathbf{A}_0^0 = \mathbf{A}_0^1$ and Steiner tree $T_1 = \emptyset$. No more row reduction is needed.

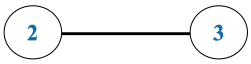


(e) The intuition behind the second reduction step.

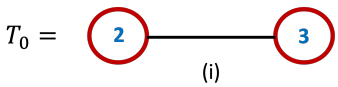
Figure 11: Illustrate the second reduction step.

$$\mathbf{A}_0 = \begin{array}{c} \\ 3 \\ 2 \\ \text{Candidate columns} \end{array} \begin{array}{c} \begin{array}{cc} 3' & 0' \end{array} \\ \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \\ \begin{array}{cc} & 2 \end{array} \end{array} \begin{array}{c} \text{Candidate rows} \\ 2 \\ 1 \\ \\ \end{array}$$

(a) For the third reduction step, the pivot row and column are highlighted in red.



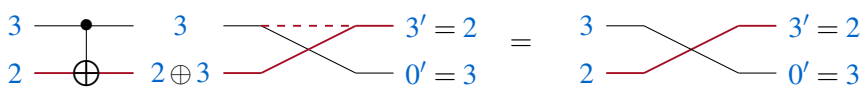
(b) G is the updated connectivity graph after removing vertex 1.



(c) Steiner tree T_0 is constructed for the column reduction. Vertex 2 is the root and vertex 3 is the leaf. $S = \{2,3\}$. The traversals of T_0 return a row operation R(2,3).

$$\mathbf{A}_0 \xrightarrow[R(2,3)]{\text{(i)}} \begin{array}{c} \\ 3 \\ 2 \end{array} \begin{array}{c} \begin{array}{cc} 3' & 0' \end{array} \\ \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \end{array} = \mathbf{A}_0^0 = \mathbf{A}_0^1.$$

(d) After the row operation, column 3 is reduced to e_3 and $\mathbf{A}_0^0$ is the evolved parity matrix. Since row 2 is also reduced to $e_0^\top$, $\mathbf{A}_0^0 = \mathbf{A}_0^1$ and Steiner tree $T_1 = \emptyset$. No more row reduction is needed.



(e) The intuition behind the third reduction step.

Figure 12: Illustrate the third reduction step.

2.4 A Primer for Noise-Aware CNOT Circuit Routing

In NISQ architectures, it is inevitable to have several error sources during the execution of a quantum circuit. Therefore, it is important to reduce their effects as much as possible. **Noise-aware CNOT circuit routing** accounts for both the CNOT gate error rates and the constraint on the NN interactions. It takes a parity matrix and an undirected edge-weighted connected graph as input. It outputs a synthesized circuit composed of allowed CNOT operations, with improved reliability.

Let ρ denote the density matrix describing the initial state of a quantum system. Let n denote the number of qubits in a quantum system, and $t = 2^n$ denote the dimension of its state space. In Section 2.4.1, we review a noisy quantum channel and its **superoperator representation**. In Section 2.4.2, we introduce a metric called the **average gate fidelity**. Compared to the CNOT count and the accumulated CNOT gate error rate, it accurately quantifies the reliability of executing a CNOT circuit on NISQ hardware.

2.4.1 Quantum Channel in Superoperator Representation

Definition 2.12. *A **quantum channel** $\mathcal{E}$ is a completely positive trace-preserving map between spaces of operators. Let $\{M_k\}$ be a set of its Kraus operators, $\sum_k M_k^\dagger M_k = I$. The action of $\mathcal{E}$ on ρ can be expressed in terms of the Kraus decomposition,*

$$\mathcal{E}(\rho) = \sum_k M_k \rho M_k^\dagger.$$

We use a quantum channel to describe the evolution of a quantum state. It provides a convenient mathematical framework for us to characterize noise in a system. We start by representing a single-qubit noisy channel $\mathcal{E}$ using the Kraus decomposition. Without loss of generality, consider a single-qubit Pauli channel [9].

Definition 2.13. *Let $\mathcal{B}_1 = \{I, X, Y, Z\}$ be a 1-qubit Pauli basis. For $n \in \mathbb{N}^{\neq 0}$, let $\mathcal{B}_n$ be an n-qubit Pauli basis.*

$$\mathcal{B}_n = \left\{ \bigotimes_{j=0}^{n-1} B_j;\ B_j \in \mathcal{B}_1 \right\}.$$

Definition 2.14. *For $n \in \mathbb{N}^{\neq 0}$, let $\mathcal{P}_n$ be the n-qubit Pauli group. For $j \in \mathbb{Z}_n$, P_j denotes the single-qubit Pauli operator acting on qubit j.*

$$\mathcal{P}_n = \left\{ \bigotimes_{j=0}^{n-1} i^c P_j;\ \ P_j \in \mathcal{B}_1,\ \ c \in \mathbb{Z}_4 \right\}.$$

Lemma 2.4. *For any $Q \in \mathcal{P}_n \setminus \{i^c I;\ \ c \in \mathbb{Z}_4\}$, $\mathsf{Tr}[Q] = 0$.*

Proof. By Definition 2.14, $Q = \bigotimes_{j=0}^{n-1} i^c P_j$, $P_j \in \mathcal{B}_1$, $c \in \mathbb{Z}_4$. Since $Q \notin \{i^c I;\ \ c \in \mathbb{Z}_4\}$, there exists some $k \in \mathbb{Z}_n$ such that $P_k \neq I$. Then, $\mathsf{Tr}[P_k] = 0$ implies that

$$\mathsf{Tr}(Q) = \mathsf{Tr}\left[\bigotimes_{j=0}^{n-1} i^c P_j \right] = i^c \prod_{j=0}^{n-1} \mathsf{Tr}[P_j] = 0.$$

□

Definition 2.15. *For $k \in \mathbb{Z}_4$, $E_k \in \mathcal{B}_1$ with $E_0 = I,\ \ E_1 = X,\ \ E_2 = Y,\ \ E_3 = Z$. P_k is the probability distribution of E_k. $0 \leq P_k \leq 1$ and $\sum P_k = 1$. M_k is the Kraus operator such that*

$$M_k = \sqrt{P_k} E_k, \qquad \sum_{k=0}^{3} M_k^\dagger M_k = I.$$

A single-qubit Pauli channel $\mathcal{E}$ is defined as

$$\mathcal{E}(\rho) = \sum_{k=0}^{3} P_k E_k \rho E_k^\dagger. \tag{8}$$

Remark 2.3. *When $P_1 = P_2 = P_3$, $\mathcal{E}$ is called the **depolarizing channel**.*

Definition 2.16. *For $n \in \mathbb{N}^{\neq 0}$, $k \in \mathbb{Z}_{4^n}$, $E_k \in \mathcal{B}_n$. P_k is the probability distribution of E_k. $0 \leq P_k \leq 1$ and $\sum P_k = 1$. M_k is the Kraus operator such that*

$$M_k = \sqrt{P_k} E_k, \qquad \sum_{k=0}^{4^n-1} M_k^\dagger M_k = I.$$

An n-qubit Pauli channel $\mathcal{E}$ is defined as

$$\mathcal{E}(\rho) = \sum_{k=0}^{4^n-1} P_k E_k \rho E_k^\dagger. \tag{9}$$

Definition 2.17. *For $m \in \mathbb{N}^{\neq 0}$, let A be an $m \times m$ matrix.*

$$A = [a_{i,j}] = \begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,m-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,m-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m-1,0} & a_{m-1,1} & \cdots & a_{m-1,m-1} \end{bmatrix}, \qquad i, j \in \mathbb{Z}_m.$$

*$a_{i,j}$ denotes the matrix element on row i and column j. $|A\rangle\rangle$ is a **column-vectorized matrix** obtained from stacking each column of A on top of one another.*

$$|A\rangle\rangle := (a_{0,0}\ a_{1,0}\ \ldots\ a_{m-1,0}\ a_{0,1}\ a_{1,1}\ \ldots\ a_{m-1,1}\ \ldots\ a_{0,m-1}\ a_{1,m-1}\ \ldots\ a_{m-1,m-1})^\top.$$

Vectorization converts an operator to a vector. It is linear in that for any two matrices A and B of the same dimension,

$$|A+B\rangle\rangle = |A\rangle\rangle + |B\rangle\rangle.$$

With the Kronecker product, matrix vectorization has a convenient property and we will use it to derive the **superoperator representation** for an arbitrary linear operator.

Lemma 2.5 ([18])**.** *Consider matrices A, B, and C with compatible dimensions for matrix multiplications,*

$$|ABC\rangle\rangle = (C^\top \otimes A)|B\rangle\rangle.$$

Definition 2.18. *Let $\mathcal{A}$ be a quantum operation acting on a density matrix ρ. ρ is of dimension $t \times t$, $t \in \mathbb{N}^{\neq 0}$. The superoperator representation $S_\mathcal{A}$ is given alongside the column-vectorized density matrix $|\rho\rangle\rangle$ as*

$$|\mathcal{A}(\rho)\rangle\rangle = S_\mathcal{A}|\rho\rangle\rangle.$$

$S_\mathcal{A}$ is of dimension $t^2 \times t^2$.

Lemma 2.6. *Let $\mathcal{A}$ be a quantum operation acting on a density matrix ρ. When $\mathcal{A}$ is unitary, let A be its unitary matrix representation. Then $S_\mathcal{A} = A^* \otimes A$.*

Proof. Since $\mathcal{A}$ is unitary, $\mathcal{A}(\rho) = A\rho A^\dagger$. By Lemma 2.5, $|\mathcal{A}(\rho)\rangle\rangle = |A\rho A^\dagger\rangle\rangle = ((A^\dagger)^\top \otimes A)|\rho\rangle\rangle = A^* \otimes A|\rho\rangle\rangle$. By Definition 2.18, $S_\mathcal{A} = A^* \otimes A$. □

Lemma 2.7. *Let $\mathcal{A}$ be a quantum operation acting on a density matrix ρ. When $\mathcal{A}$ is not unitary, let $\{M_k\}$ be a set of its Kraus operators. Then $S_\mathcal{A} = \sum_k M_k^* \otimes M_k$.*

Proof. Since $\mathcal{A}$ is not unitary, $\mathcal{A}(\rho) = \sum_k M_k \rho M_k^\dagger$, with $\sum_k M_k^\dagger M_k = I$. By linearity and Lemma 2.5,

$$|\mathcal{A}(\rho)\rangle\rangle = |\sum_k M_k \rho M_k^\dagger\rangle\rangle = \sum_k |M_k \rho M_k^\dagger\rangle\rangle = \sum_k M_k^* \otimes M_k |\rho\rangle\rangle. \tag{10}$$

By Definition 2.18, $S_\mathcal{A} = \sum_k M_k^* \otimes M_k$. □

Corollary 2.1. For E_k, P_k, and M_k in Definition 2.16, let $S_{E_k} = E_k^* \otimes E_k$. Then

$$S_\mathcal{E} = \sum_{k=0}^{4^n-1} P_k S_{E_k}.$$

Proof. By Lemma 2.7 and definition 2.16,

$$S_\mathcal{E} = \sum_{k=0}^{4^n-1} M_k^* \otimes M_k = \sum_{k=0}^{4^n-1} P_k (E_k^* \otimes E_k).$$

Since $E_k \in \mathcal{B}_n$ is unitary, by Lemma 2.6, $S_{E_k} = E_k^* \otimes E_k$. Therefore, $S_\mathcal{E} = \sum_{k=0}^{4^n-1} P_k S_{E_k}$.

□

Lemma 2.8. *For E_j, E_k in Definition 2.16, $t = 2^n$, and $n \in \mathbb{N}^{\neq 0}$, $Tr[E_j E_k] = t\delta_{jk}$.*

Proof. When $j = k$, $E_j E_k = E_j^2 = I$. Since I is an identity matrix of dimension $t \times t$, $\mathsf{Tr}[E_i E_j] = \mathsf{Tr}[I] = t$. Otherwise, $E_j E_k \notin \{i^c I;\ c \in \mathbb{Z}_4\}$. By Lemma 2.4, $\mathsf{Tr}[E_j E_k] = 0$. □

Lemma 2.9. *For E_k in Definition 2.16, $E_k \neq I$ and $t = 2^n$. Let $E_0 = I$. Then $Tr[S_{E_0}] = t^2$. For $k > 0$, $Tr[S_{E_k}] = 0$.*

Proof. By Definition 2.18, S_{E_0} is an identity matrix of dimension $t^2 \times t^2$. By direct computation, $\mathsf{Tr}[S_{E_0}] = \mathsf{Tr}[I] = t^2$. For $k > 0$, since $E_k \neq I$, by Lemmas 2.4 and 2.6, $\mathsf{Tr}[S_{E_k}] = \mathsf{Tr}[E_k^* \otimes E_k] = \mathsf{Tr}[E_k]^* \mathsf{Tr}[E_k] = 0$. □

Lemma 2.10. *Let $\mathcal{E}_0$ and $\mathcal{E}_1$ be two quantum channels with respective sets of Kraus operators $\{M_k;\ 0 \leq k < n_0,\ n_0 \in \mathbb{N}^{\neq 0}\}$ and $\{N_\ell;\ 0 \leq \ell < n_1,\ n_1 \in \mathbb{N}^{\neq 0}\}$. $\mathcal{E}_0$ and $\mathcal{E}_1$ have compatible dimensions. $\sum M_k^\dagger M_k = \sum N_\ell^\dagger N_\ell = I$. Then $S_{\mathcal{E}_1 \circ \mathcal{E}_0} = S_{\mathcal{E}_1} S_{\mathcal{E}_0}$.*

Proof. Let $\mathcal{K} = \{N_\ell M_k;\ 0 \leq k < n_0,\ 0 \leq \ell < n_1,\ n_0, n_1 \in \mathbb{N}^{\neq 0}\}$. $\mathcal{K}$ satisfies the completeness equation since

$$\sum_{\ell,k} (N_\ell M_k)^\dagger (N_\ell M_k) = \sum_{\ell,k} (M_k^\dagger N_\ell^\dagger)(N_\ell M_k) = \sum_k M_k^\dagger \Big(\sum_\ell N_\ell^\dagger N_\ell\Big) M_k = \sum_k M_k^\dagger M_k = I.$$

Hence, $\mathcal{K}$ is a set of Kraus operators for $\mathcal{E}_1 \circ \mathcal{E}_0$. By Lemma 2.7, $S_{\mathcal{E}_1 \circ \mathcal{E}_0} = \sum_{\ell,k} (N_\ell M_k)^* \otimes (N_\ell M_k)$. Using the linearity and cyclicity of trace with the property that $\mathsf{Tr}[A \otimes B] = \mathsf{Tr}[A]\mathsf{Tr}[B]$,

$$\begin{aligned}
S_{\mathcal{E}_1 \circ \mathcal{E}_0} &= \sum_{\ell,k} (N_\ell M_k)^* \otimes (N_\ell M_k) \\
&= \sum_{\ell,k} (N_\ell^* M_k^*) \otimes (N_\ell M_k) \\
&= \sum_{\ell,k} (N_\ell^* \otimes N_\ell)(M_k^* \otimes M_k) \\
&= \Big(\sum_\ell (N_\ell^* \otimes N_\ell)\Big)\Big(\sum_k (M_k^* \otimes M_k)\Big) = S_{\mathcal{E}_1} S_{\mathcal{E}_0}.
\end{aligned}$$

□

Lemma 2.11. *Let $\mathcal{E}_0$ and $\mathcal{E}_1$ be two quantum channels with respective sets of Kraus operators $\{M_k;\ 0 \le k < n_0,\ n_0 \in \mathbb{N}^{\neq 0}\}$ and $\{N_\ell;\ 0 \le \ell < n_1,\ n_1 \in \mathbb{N}^{\neq 0}\}$. $\mathcal{E}_0$ and $\mathcal{E}_1$ have compatible dimensions. $\sum M_k^\dagger M_k = I_0$ and $\sum N_\ell^\dagger N_\ell = I_1$, I_0 and I_1 are identity matrices.*

$$Tr[S_{\mathcal{E}_0 \otimes \mathcal{E}_1}] = Tr[S_{\mathcal{E}_0}]\, Tr[S_{\mathcal{E}_1}].$$

Proof. Let $\mathcal{K} = \{M_k \otimes N_\ell;\ 0 \le k < n_0,\ 0 \le \ell < n_1,\ n_0, n_1 \in \mathbb{N}^{\neq 0}\}$. $\mathcal{K}$ satisfies the completeness equation since

$$\sum_{k,\ell} (M_k \otimes N_\ell)^\dagger (M_k \otimes N_\ell) = \sum_{k,\ell} (M_k^\dagger \otimes N_\ell^\dagger)(M_k \otimes N_\ell) = \sum_{k,\ell} (M_k^\dagger M_k) \otimes (N_\ell^\dagger N_\ell) = \left(\sum_k M_k^\dagger M_k\right) \otimes \left(\sum_\ell N_\ell^\dagger N_\ell\right) = I.$$

Hence, $\mathcal{K}$ is a set of Kraus operators for $\mathcal{E}_0 \otimes \mathcal{E}_1$. Using the linearity of trace and the property that $\mathsf{Tr}[A \otimes B] = \mathsf{Tr}[A]\mathsf{Tr}[B]$,

$$\begin{aligned}
\mathsf{Tr}[S_{\mathcal{E}_0 \otimes \mathcal{E}_1}] &= \mathsf{Tr}\left[\sum_{k,\ell} (M_k \otimes N_\ell)^* \otimes (M_k \otimes N_\ell)\right] \\
&= \sum_{k,\ell} \mathsf{Tr}\left[(M_k^* \otimes N_\ell^*) \otimes (M_k \otimes N_\ell)\right] \\
&= \sum_{k,\ell} \mathsf{Tr}[M_k^*]\mathsf{Tr}[N_\ell^*]\mathsf{Tr}[M_k]\mathsf{Tr}[N_\ell] \\
&= \sum_{k,\ell} \mathsf{Tr}[M_k^*]\mathsf{Tr}[M_k]\mathsf{Tr}[N_\ell^*]\mathsf{Tr}[N_\ell] \\
&= \sum_{k,\ell} \mathsf{Tr}[M_k^* \otimes M_k]\mathsf{Tr}[N_\ell^* \otimes N_\ell] \\
&= \sum_{k} \mathsf{Tr}[M_k^* \otimes M_k] \sum_{\ell} \mathsf{Tr}[N_\ell^* \otimes N_\ell] \\
&= \mathsf{Tr}\left[\sum_k M_k^* \otimes M_k\right] \mathsf{Tr}\left[\sum_\ell N_\ell^* \otimes N_\ell\right] = \mathsf{Tr}[S_{\mathcal{E}_0}]\mathsf{Tr}[S_{\mathcal{E}_1}].
\end{aligned}$$

□

2.4.2 Average Gate Fidelity

In an ideal world, a quantum algorithm can be precisely implemented by a sequence of carefully selected gates, and the state evolution is described by a unitary transformation. Let $\mathcal{U}$ be the linear map describing its transformation and U be its unitary matrix representation. Then $\mathcal{U}(\rho) = U\rho U^\dagger$ denotes the ideal evolved state of ρ. In reality, quantum operations are prone to errors and $\mathcal{U}$ is approximated by a noisy quantum channel $\mathcal{E}$. By Definition 2.12, $\mathcal{E}(\rho)$ denotes the actual evolved state of ρ,

$$\mathcal{E}(\rho) = \sum_k M_k \rho M_k^\dagger. \tag{11}$$

When $\mathcal{E}$ is "close" to $\mathcal{U}$, the resulting quantum evolution is closely aligned with the desired algorithm. Let σ be the density operator describing the quantum state of another physical system. Recall that in [43], the **state fidelity** of ρ and σ is defined as $F(\rho, \sigma)$,

$$F(\rho, \sigma) = \left(Tr\left[\sqrt{\sqrt{\rho}\ \sigma \sqrt{\rho}}\right]\right)^2. \tag{12}$$

Let $\mathcal{E}(\rho)$ denote the final state of ρ after the action of $\mathcal{E}$. The **gate fidelity**, $F_{\mathcal{U},\mathcal{E}}(\rho)$, is defined as state fidelity of $\mathcal{E}(\rho)$ and $\mathcal{U}(\rho)$,

$$F_{\mathcal{U},\mathcal{E}}(\rho) = \left(Tr\left[\sqrt{\sqrt{\mathcal{U}(\rho)}\ \mathcal{E}(\rho) \sqrt{\mathcal{U}(\rho)}}\right]\right)^2. \tag{13}$$

When the input state is a pure state, $\rho = |\psi\rangle\langle\psi|$ where $|\psi\rangle$ is the state vector. Using the cyclic property of trace, Equation (13) is reduced to Equation (14).

$$F_{\mathcal{U},\mathcal{E}}(|\psi\rangle\langle\psi|) = \left(Tr\left[\sqrt{U|\psi\rangle\langle\psi|U^\dagger \mathcal{E}(|\psi\rangle\langle\psi|)}\right]\right)^2 = \langle\psi|U^\dagger \mathcal{E}(|\psi\rangle\langle\psi|)U|\psi\rangle. \tag{14}$$

In Equation (15), we obtain the average gate fidelity by integrating over all pure input states [12]. It provides a concise measure of error independent of the input state. This is more accurate than merely counting the number of CNOT gates [26, 25, 48, 39] or calculating the sum of CNOT gate error rates [71]. Hence, we use it as a cost function to gauge the proximity between the dynamic evolution and the desired evolution.

$$F_{avg}(\mathcal{E},\mathcal{U}) = \int d\psi F_{\mathcal{U},\mathcal{E}}(|\psi\rangle\langle\psi|) = \int d\psi \langle\psi|U^\dagger \mathcal{E}(|\psi\rangle\langle\psi|)U|\psi\rangle. \tag{15}$$

Let p be the error rate of a noisy quantum channel $\mathcal{E}$ whose ideal operation is $\mathcal{U}$. Then

$$p = 1 - F_{avg}(\mathcal{E},\mathcal{U}). \tag{16}$$

Next, we show that the average gate fidelity assesses how closely $\mathcal{E}$ approximates $\mathcal{U}$ independent of the input states. We start by simplifying Equation (15) and let $\mathcal{E}' = \mathcal{U}^{-1} \circ \mathcal{E}$. Accordingly, the modified set of Kraus operators is $\{M'_k;\ M'_k = U^\dagger M_k\}$. To compute $F_{avg}(\mathcal{E},\mathcal{U})$, it is equivalent to calculate $F_{avg}(\mathcal{E}',\mathcal{I})$.

Lemma 2.12.

$$F_{avg}(\mathcal{E},\mathcal{U}) = F_{avg}(\mathcal{E}',\mathcal{I}).$$

Proof. Note that $\sum M_k^\dagger M_k = I$ and U is unitary. Then $\{M'_k;\ M'_k = U^\dagger M_k\}$ is a valid set of Kraus operators, since

$$\sum {M_k'}^\dagger M'_k = \sum (U^\dagger M_k)^\dagger (U^\dagger M_k) = \sum M_k^\dagger (UU^\dagger) M_k = \sum M_k^\dagger M_k = I.$$

Based on Equations (11) and (15), we have

$$\begin{aligned} F_{avg}(\mathcal{E},\mathcal{U}) &= \int d\psi \langle\psi|U^\dagger \left(\sum M_k|\psi\rangle\langle\psi|M_k^\dagger\right) U|\psi\rangle \\ &= \int d\psi \langle\psi| \left(\sum U^\dagger M_k|\psi\rangle\langle\psi|M_k^\dagger U\right) |\psi\rangle \\ &= \int d\psi \langle\psi| \left(\sum M'_k|\psi\rangle\langle\psi|{M_k}'^\dagger\right) |\psi\rangle \\ &= \int d\psi \langle\psi|\mathcal{E}'(|\psi\rangle\langle\psi|)|\psi\rangle = F_{avg}(\mathcal{E}',\mathcal{I}). \end{aligned}$$

□

For brevity, we write $F_{avg}(\mathcal{E}')$ for $F_{avg}(\mathcal{E}',\mathcal{I})$. [32, 50] provide an alternative expression for the average gate fidelity, as shown in Equation (17). $F_{pro}(\mathcal{E}',\mathcal{I})$ is called the **process fidelity** (a.k.a., the **entanglement fidelity**) and it gauges the overlap between ρ before and after the application of $\mathcal{E}'$. It describes how well the quantum information in a system and the entanglement with other systems are preserved [62].

$$F_{avg}(\mathcal{E}') = \frac{tF_{pro}(\mathcal{E}') + 1}{t+1}. \tag{17}$$

Lemma 2.13.

$$F_{pro}(\mathcal{E}') = \frac{Tr[S_{\mathcal{E}'}]}{t^2}.$$

Proof details could be found in Appendix A. Combining Lemmas 2.12 and 2.13, we have

$$F_{avg}(\mathcal{E}') = \frac{tF_{pro}(\mathcal{E}')+1}{t+1} = \frac{\mathsf{Tr}[S_{\mathcal{E}'}]}{t(t+1)} + \frac{1}{t+1}. \tag{18}$$

Finally, by using Equations (16) and (18), we derive an equality that will be used in Section 3.2.

$$F_{avg}(\mathcal{E}) = 1 - p = \frac{\mathsf{Tr}[S_{\mathcal{E}}]}{t(t+1)} + \frac{1}{t+1}. \tag{19}$$

3 Quantify the Reliability of a Noisy CNOT Circuit

To gauge the quality of a noisy CNOT circuit, we define a cost function by approximating

Definition 3.1. *Let $\mathcal{E}$ be an error channel. The error probability of $\mathcal{E}$ is $Prob(\mathcal{E})$,*

$$Prob(\mathcal{E}) = 1 - F_{avg}(\mathcal{E}).$$

Since the size of the superoperator in Equation (19) scales exponentially with the number of qubits in the system, exactly computing $F_{avg}(\mathcal{E})$ demands a substantial amount of computational resource. In light of this, we model a noisy CNOT circuit using both the **parallel error channel** and the **consecutive error channel**. Based on the NISQ specifics (the number of physical qubits, and the range of CNOT gate error rates), we find a tight approximation for the average gate fidelity and use it as our cost function. The higher it is, the worse the performance of a noise-aware CNOT circuit routing algorithm.

Definition 3.2. *Let $\mathbf{C}$ be an n-qubit quantum circuit with m noisy gates. Numbering each gate from left to right, $Prob(\mathcal{E}_i)$ is the error probability of the i-th noisy channel $\mathcal{E}_i$. Let $Cost(\mathbf{C})$ be a cost function of $\mathbf{C}$.*

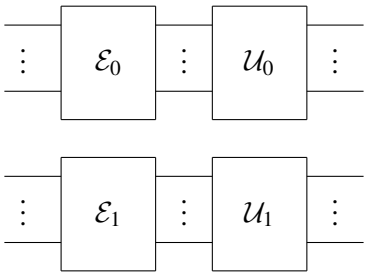
$$Cost(\mathbf{C}) = 1 - \prod_{i=0}^{m-1}(1 - Prob(\mathcal{E}_i)).$$

In Sections 3.1 and 3.2, we characterize a noisy parallel and consecutive error channels using the superoperator representation and calculate their respective average gate fidelity. In Section 3.3, we combine both to approximate the average gate fidelity of a noisy CNOT circuit and formally define its cost function based on Definition 3.2.

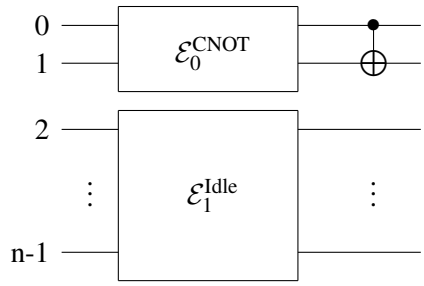
3.1 Parallel Error Channel

A parallel error channel $\mathcal{E}_q$ is vertically composed of channels with arbitrary input and output dimensions. Let t be its input dimension. As shown in Figure 13a, $\mathcal{E}_0$ and $\mathcal{E}_1$ are two error channels followed by two ideal operations $\mathcal{U}_0$ and $\mathcal{U}_1$. U_0 and U_1 are their respective unitary matrix representations. Let p_0 and p_1 be the error rates of $\mathcal{E}_0$ and $\mathcal{E}_1$. Let d_0 and d_1 be the respectively input dimensions and $k \in \{0,1\}$. $t = d_0 d_1$. By Equation (19), we can express $\mathsf{Tr}[S_{\mathcal{E}_k}]$ in terms of d_k and p_k.

$$\frac{\mathsf{Tr}[S_{\mathcal{E}_k}]}{d_k^2} = 1 - \frac{d_k+1}{d_k}p_k. \tag{20}$$



(a) The error channel $\mathcal{E}_q$ is vertically composed of two error channels $\mathcal{E}_0$ and $\mathcal{E}_1$, followed the respective ideal operations $\mathcal{U}_0$ and $\mathcal{U}_1$. $\mathcal{E}_0$ has an error rate p_0 and $\mathcal{E}_1$ has an error rate p_1. Their input dimensions are d_0 and d_1.



(b) Without loss of generality, suppose that $\mathcal{E}_q$ is composed of two parallel channels, $\mathcal{E}_0^{\text{CNOT}}$ and $\mathcal{E}_1^{\text{Idle}}$. $\mathcal{E}_0^{\text{CNOT}}$ is the error channel of a noisy CNOT gate. Its error rate is p_0. $\mathcal{E}_1^{\text{Idle}}$ is the error channel of $n-2$ idle qubits. Its error rate is p_1.

Figure 13: The parallel error channels.

Lemma 3.1. *Let $\mathcal{E}_q$ be an n-qubit channel that is vertically composed of two noisy channels $\mathcal{E}_0$ and $\mathcal{E}_1$, with input dimension d_0 and d_1, error probability p_0 and p_1. $\mathcal{E}_q = \mathcal{E}_0 \otimes \mathcal{E}_1$, $d_q = d_0 d_1 = 2^n$. d_q is the input dimension of $\mathcal{E}_q$. Then*

$$F_{ave}(\mathcal{E}_q) = 1 - p_0 - p_1 + p_0 p_1 + \frac{(1-d_1)p_0 + (1-d_0)p_1 + (d_0+d_1)p_0 p_1}{d_0 d_1 + 1}.$$

Proof. Applying Lemma 2.11 with Equations (19) and (20), we have

$$\begin{aligned} F_{avg}(\mathcal{E}_q) = F_{avg}(\mathcal{E}_0 \otimes \mathcal{E}_1) &= \frac{\mathsf{Tr}[S_{\mathcal{E}_0 \otimes \mathcal{E}_1}]}{(d_0 d_1)^2} \times \frac{d_0 d_1}{d_0 d_1 + 1} + \frac{1}{d_0 d_1 + 1} \\ &= \frac{\mathsf{Tr}[S_{\mathcal{E}_0}]}{d_0^2} \frac{\mathsf{Tr}[S_{\mathcal{E}_1}]}{d_1^2} \times \frac{d_0 d_1}{d_0 d_1 + 1} + \frac{1}{d_0 d_1 + 1} \\ &= 1 - p_0 - p_1 + p_0 p_1 + \frac{(1-d_1)p_0 + (1-d_0)p_1 + (d_0+d_1)p_0 p_1}{d_0 d_1 + 1} \end{aligned} \tag{21}$$

Technical details can be found in Appendix A. □

3.2 Consecutive Error Channel

A consecutive error channel $\mathcal{E}_c$ is horizontally composed of channels with compatible input and output dimensions. Let t be its input dimension. In Figure 14, $\mathcal{E}_0$ and $\mathcal{E}_1$ are two error channels followed by two ideal operations $\mathcal{U}_0$ and $\mathcal{U}_1$. U_0 and U_1 are their respective unitary matrix representations. Let p_0 and p_1 be the error rates of $\mathcal{E}_0$ and $\mathcal{E}_1$. Let d_0 and d_1 be the respective input dimensions and $k \in \mathbb{Z}_2$. $t = d_0 = d_1$.

$$\mathcal{E}_c = \mathcal{U}_1 \circ \mathcal{E}_1 \circ \mathcal{U}_0 \circ \mathcal{E}_0.$$

Here $\circ$ denotes the composition of linear maps.

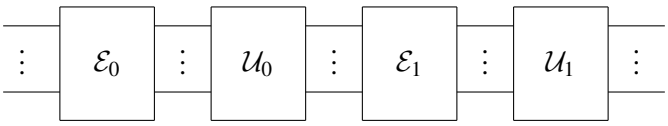


Figure 14: The error channel $\mathcal{E}_c$ is horizontally composed of two error channels $\mathcal{E}_0$ and $\mathcal{E}_1$, followed the respective ideal operations $\mathcal{U}_0$ and $\mathcal{U}_1$.

Lemma 3.2. *When two noisy Clifford channels are composed horizontally, the identity in Figure 15 holds.*

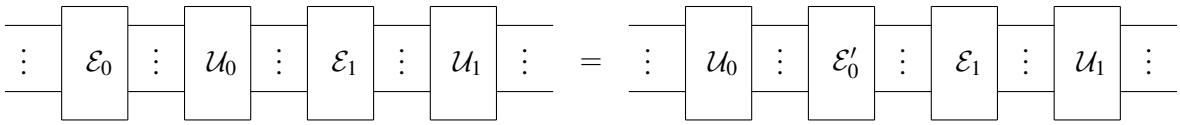


Figure 15: $\mathcal{E}_c$ is composed of two noisy Clifford gates. Let n be the number of qubits. p_0 and p_1 are the error rates of $\mathcal{E}_0$ and $\mathcal{E}_1$. Their input dimensions are identical, $d_0 = d_1 = 2^n$. U_0 and U_1 are the respective unitary matrix representations of $\mathcal{U}_0$ and $\mathcal{U}_1$. $U_0, U_1 \in \mathcal{C}_n$.

Proof. Let ρ be the density matrix describing the initial state of the quantum system in Figure 15. It is sufficient to show that

$$\mathcal{U}_0 \circ \mathcal{E}_0(\rho) = \mathcal{E}'_0 \circ \mathcal{U}_0(\rho).$$

By Definition 2.16, let $\{M_k\}$ be a set of Kraus operators of $\mathcal{E}_0$ such that $E_k \in \mathcal{B}_n$,

$$M_k = \sqrt{P_k}E_k, \qquad \sum M_k^\dagger M_k = I, \qquad \sum P_k = 1, \qquad 0 \le P_k \le 1.$$

Since $U_0 \in \mathcal{C}_n$ and $E_k \in \mathcal{P}_n$, $U_0 E_k U_0^\dagger = E'_k$, $E'_k \in \mathcal{P}_n$. Let $M'_K = \sqrt{P_k}E'_k$. It follows that

$$U_0 E_k = E'_k U_0, \qquad U_0 M_k = M'_k U_0 \tag{22}$$

Since $\sum P_k E_k^\dagger E_k = I$ and U_0 is unitary, to show that $\forall k$, $M'_k = \sqrt{P_k}E'_k$ are valid Kraus operators, we have

$$\sum M'^\dagger_k M'_k = \sum P_i E'^\dagger_k E'_k = \sum P_i \left(U_0 E_k^\dagger U_0^\dagger\right)\left(U_0 E_k U_0^\dagger\right) = \sum P_i \left(U_0 E_k^\dagger E_k U_0^\dagger\right) = U_0 \left(\sum P_k E_k^\dagger E_k\right) U_0^\dagger = U_0 I U_0^\dagger = I.$$

Hence, there exists a channel $\mathcal{E}'_0$ over E'_k such that $\mathcal{E}'_0(\rho) = \sum M'_k \rho M'^\dagger_k$. Therefore,

$$\begin{aligned} LHS = \mathcal{U}_0 \circ \mathcal{E}_0(\rho) &= \mathcal{U}_0\left(\mathcal{E}_0(\rho)\right) = \sum \mathcal{U}_0\left(M_k \rho M_k^\dagger\right) \\ &= \sum U_0\left(M_k \rho M_k^\dagger\right) U_0^\dagger \\ &= \sum M'_k U_0 \rho U_0^\dagger M'^\dagger_k \\ &= \sum M'_k\left(\mathcal{U}_0(\rho)\right) M'^\dagger_k \\ &= \mathcal{E}'_0 \circ \mathcal{U}_0(\rho) = RHS \end{aligned}$$

□

Remark 3.1. *Let $m \in \mathbb{N}$, Lemma 3.2 generalizes to a horizontal composition of m noisy Clifford channels. This can be proved by induction on the number of CNOT gates.*

Corollary 3.1. Up to Clifford conjugation, the composite error channel in Figure 14 is $\mathcal{E}_c = \mathcal{E}_1 \circ \mathcal{E}_0$. Moreover,

$$F_{avg}(\mathcal{E}_c) = \frac{\mathsf{Tr}[S_{\mathcal{E}_1} S_{\mathcal{E}_0}]}{t(t+1)} + \frac{1}{t+1}$$

Proof. By Equation (19) and lemma 2.10, we can calculate the average gate fidelity $\mathcal{E}_c$ as

$$F_{avg}(\mathcal{E}_c) = F_{avg}(\mathcal{E}_1 \circ \mathcal{E}_0) = \frac{\mathsf{Tr}[S_{\mathcal{E}_1 \circ \mathcal{E}_0}]}{t^2} \times \frac{t}{t+1} + \frac{1}{t+1} = \frac{\mathsf{Tr}[S_{\mathcal{E}_1} S_{\mathcal{E}_0}]}{t(t+1)} + \frac{1}{t+1}.$$

□

Lemma 3.3. *Let I be an identity matrix of size $t^2 \times t^2$. For $k \in \mathbb{Z}_2$, $A_k = I - S_{\mathcal{E}_k}$.*

$$Tr[S_{\mathcal{E}_1} S_{\mathcal{E}_0}] = Tr[S_{\mathcal{E}_0}] + Tr[S_{\mathcal{E}_1}] - t^2 + Tr[A_1 A_0]$$

Proof. By the linearity of trace,

$$\mathsf{Tr}[A_k] = \mathsf{Tr}[I - S_{\mathcal{E}_k}] = \mathsf{Tr}[I] - \mathsf{Tr}[S_{\mathcal{E}_k}] = t^2 - \mathsf{Tr}[S_{\mathcal{E}_k}].$$

It follows that

$$\begin{aligned}
\mathsf{Tr}[S_{\mathcal{E}_1} S_{\mathcal{E}_0}] &= \mathsf{Tr}[(I - A_1)(I - A_0)] \\
&= \mathsf{Tr}[I - A_1 - A_0 + A_1 A_0] \\
&= t^2 - \mathsf{Tr}[A_1] - \mathsf{Tr}[A_0] + \mathsf{Tr}[A_1 A_0] \\
&= \mathsf{Tr}[S_{\mathcal{E}_0}] + \mathsf{Tr}[S_{\mathcal{E}_1}] - t^2 + \mathsf{Tr}[A_1 A_0].
\end{aligned}$$

□

Lemma 3.4. *Let $t = 2^n$ be the input dimension of $\mathcal{E}_c$. For $k \in \mathbb{Z}_2$, $d_k = t$ is the input dimension of an error channel $\mathcal{E}_k$ with error probability p_k. If $\mathcal{E}_c = \mathcal{E}_1 \circ \mathcal{E}_0$ and $A_k = I - S_{\mathcal{E}_k}$,*

$$(t+1)^2 p_0 p_1 \leq Tr[A_1 A_0] \leq 2(t+1)^2 p_0 p_1.$$

Proof. Justification is detailed in Appendix A. □

Lemma 3.5. *Let $\mathcal{E}_c$ be an n-qubit channel that is horizontally composed of two noisy channels $\mathcal{E}_0$ and $\mathcal{E}_1$, with input dimensions d_0 and d_1, error probabilities p_0 and p_1. $0 \leq p_0, p_1 \leq 1$. $\mathcal{E}_c = \mathcal{E}_1 \circ \mathcal{E}_0$, $t = d_0 = d_1 = 2^n$. t is the input dimension of $\mathcal{E}_c$. Then*

$$0 \leq F_{avg}(\mathcal{E}_c) - (1-p_0)(1-p_1) \leq \left(1 + \frac{1}{2^{n-1}}\right) p_0 p_1.$$

Proof. Based on Equation (19),

$$\mathsf{Tr}[S_{\mathcal{E}_0}] = t(t+1)(1-p_0) - t, \qquad \mathsf{Tr}[S_{\mathcal{E}_1}] = t(t+1)(1-p_1) - t.$$

By Corollary 3.1 and lemma 3.3, we have

$$\begin{aligned}
F_{avg}(\mathcal{E}_c) &= \frac{\mathsf{Tr}[S_{\mathcal{E}_0}] + \mathsf{Tr}[S_{\mathcal{E}_1}] - t^2 + \mathsf{Tr}[A_1 A_0]}{t(t+1)} + \frac{1}{t+1} \\
&= \frac{t(t+1)(1-p_0) - t + t(t+1)(1-p_1) - t - t^2}{t(t+1)} + \frac{\mathsf{Tr}[A_1 A_0]}{t(t+1)} + \frac{1}{t+1} \\
&= 1 - p_0 - p_1 + \frac{\mathsf{Tr}[A_1 A_0]}{t(t+1)}.
\end{aligned} \tag{23}$$

Combining Lemma 3.4 and equation (23), we get

$$1 - p_0 - p_1 + p_0 p_1 + \frac{1}{t} p_0 p_1 \leq F_{avg}(\mathcal{E}_c) \leq 1 - p_0 - p_1 + 2 p_0 p_1 + \frac{2}{t} p_0 p_1. \tag{24}$$

Since $d = 2^n$,

$$\frac{1}{2^n} p_0 p_1 \leq \left[F_{avg}(\mathcal{E}_c) - (1 - p_0 - p_1 + p_0 p_1)\right] \leq p_0 p_1 + \frac{2}{2^n} p_0 p_1.$$

Therefore,

$$0 \leq F_{avg}(\mathcal{E}_c) - (1 - p_0 - p_1 + p_0 p_1) \leq p_0 p_1 + \frac{2}{2^n} p_0 p_1 = \left(1 + \frac{1}{2^{n-1}}\right) p_0 p_1.$$

□

3.3 Approximate Average Gate Fidelity

To simplify discussions, we assume no parallelization of quantum gates and no noise on idle qubits. Then any noisy quantum circuit can be modelled as a horizontal composition of noisy gates. Figure 16 shows an example of a noisy CNOT circuits over 4 qubits.

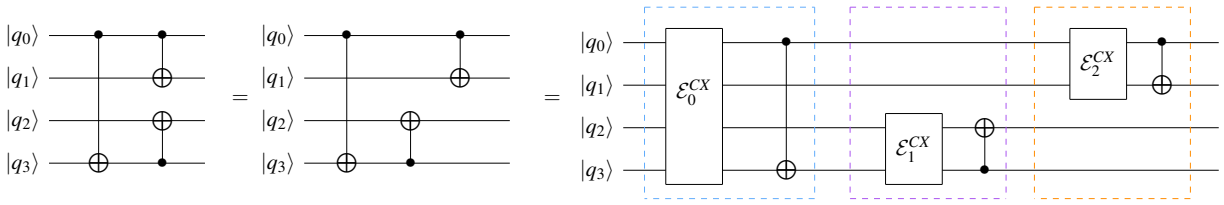


Figure 16: Any noisy CNOT circuit is horizontally composed of noisy CNOT gates when we assume no gate parallelization.

Combining the notions developed in Sections 3.1 and 3.2, Definition 3.3 specifies a cost function for a noisy CNOT circuit **C**. By Lemma 3.2, consider an equivalent circuit of **C** where the noisy channel for each CNOT gate is placed adjacent to each other. Let $\mathcal{E}_c$ be this horizontally composed noisy channel. Let m be the number of CNOT gates in the circuit.

When a channel is composed of one noisy CNOT gate When $m = 1$, based on Lemma 3.1, we can calculate the error probability of a noisy CNOT channel.

Corollary 3.2. Let $\mathcal{E}_q$ be an n-qubit error channel for a noisy CNOT gate, with error rate p and $(n-2)$ idle qubits. Then, $\mathsf{Prob}(\mathcal{E}_q) = \big(1 + \frac{2^{n-2}-1}{2^n+1}\big)p$.

Proof. Without loss of generality, suppose that $\mathcal{E}_q$ is composed of two parallel channels, $\mathcal{E}_0^{\text{CNOT}}$ and $\mathcal{E}_1^{\text{Idle}}$, as shown in Figure 13b. $\mathcal{E}_0^{\text{CNOT}}$ is the error channel of a noisy CNOT gate. Its error rate is $p_0 = p$. $\mathcal{E}_1^{\text{Idle}}$ is the error channel of $n-2$ idle qubits. Its error rate is p_1. By assumption, $p_1 = 0$. Moreover, $d_0 = 2^2 = 4$, $d_1 = 2^{n-2}$. Substituting the variables in Equation (21), we have

$$\begin{aligned} F_{avg}(\mathcal{E}_q) &= 1 - p_0 - p_1 + p_0p_1 + \frac{(1-d_1)p_0 + (1-d_0)p_1 + (d_0+d_1)p_0p_1}{d_0d_1+1} \\ &= 1 - p - 0 + 0 + \frac{(1-2^{n-2})p + 0 + 0}{2^2 \times 2^{n-2} + 1} \\ &= 1 - \big(1 + \frac{2^{n-2}-1}{2^n+1}\big)p \end{aligned}$$

By Definition 3.1, $\mathsf{Prob}(\mathcal{E}_q) = 1 - F_{avg}(\mathcal{E}_q) = \big(1 + \frac{2^{n-2}-1}{2^n+1}\big)p$. □

When a channel is composed of two noisy CNOT gates When $m = 2$, based on Corollary 3.2 and lemma 3.5, we can calculate the error probability of two horizontally composed noisy CNOT channels.

Corollary 3.3. Let **C** be an n-qubit circuit with two noisy CNOT gates. Let $\mathcal{E}_0$ and $\mathcal{E}_1$ be their respective error channels with error rates p_0 and p_1. Assume no parallelization of CNOT gates. In each channel, there are $(n-2)$ idle qubits. Let $\alpha = 1 + \frac{2^{n-2}-1}{2^n+1}$. Then,

$$0 \le F_{avg}(\mathcal{E}_c) - (1 - \mathsf{Prob}(\mathcal{E}_0))(1 - \mathsf{Prob}(\mathcal{E}_1)) \le \Big(1 + \frac{1}{2^{n-1}}\Big)\alpha^2 p_0 p_1.$$

Proof. By Corollary 3.2, the error probabilities of $\mathcal{E}_0$ and $\mathcal{E}_1$ are

$$\mathsf{Prob}(\mathcal{E}_0) = \alpha p_0, \quad \mathsf{Prob}(\mathcal{E}_1) = \alpha p_1.$$

Substituting p_0 and p_1 by $\mathsf{Prob}(\mathcal{E}_0)$ and $\mathsf{Prob}(\mathcal{E}_1)$ in Lemma 3.5, we have

$$0 \leq LHS \leq \left(1 + \frac{1}{2^{n-1}}\right)(\alpha p_0)(\alpha p_1) = \left(1 + \frac{1}{2^{n-1}}\right)\alpha^2 p_0 p_1 = RHS.$$

□

When a channel is composed of multiple noisy CNOT gates Definition 3.3 proposes a new cost function for a noisy CNOT circuit based on inspecting the approximated form of error probability in simpler cases. In Section 5.1, we compare it with the sum of CNOT error rates and the average gate fidelity by running simulations with CNOT circuits of small size.

Definition 3.3. *Let $d = 2^n$ be the input dimension of $\mathcal{E}_c$. For all $m \in \mathbb{N}$ and $k \in \mathbb{Z}_m$, $d_k = d$ is the input dimension of an error channel $\mathcal{E}_k$ consists of a noisy CNOT gate with error rate p_k. Let $\alpha = 1 + \frac{2^{n-2}-1}{2^n+1}$. Assume no parallelization of CNOT gates. In each channel, there are $(n-2)$ idle qubits. Then,*

$$\mathsf{Cost}(\mathcal{E}_c) = 1 - \prod_{i=0}^{m-1}(1 - \alpha p_i).$$

Remark 3.2. *For all $n \in \mathbb{N}$, $1 < \alpha < \frac{5}{4}$, since*

$$0 < \frac{2^{n-2}-1}{2^n+1}, \quad 1 + \frac{2^{n-2}-1}{2^n+1} < 1 + \frac{2^{n-2}}{2^n} = 1 + \frac{1}{4} = \frac{5}{4}.$$

For all $m \in \mathbb{N}$, $i \in \mathbb{Z}_m$, when $0 < p_i < 0.8$, $0 < \alpha p_i < 1$, since

$$0 < \alpha p_i < \frac{5}{4} p_i < 1.$$

Appendix C shows that in all benchmarked backends, the error rate of any CNOT gate is bounded by 0.1. By Remark 3.2, for any synthesized CNOT circuit $\mathbf{C}_{syn}$, $\mathsf{Cost}(\mathbf{C}_{syn})$ ranges between 0 and 1.

4 Noise-Aware CNOT Circuit Routing: NAPermRowCol

A **noise-aware CNOT circuit routing** algorithm maps a logical CNOT circuit $\mathbf{C}$ to a NISQ hardware. It takes $\mathbf{C}$'s parity matrix $\mathbf{A}$ and an undirected edge-weighted connected graph G as inputs. A vertex in G corresponds to a physical qubit. An edge in G represents an allowed CNOT operation on the qubits corresponding to its endpoints. Its edge weight records the CNOT gate error rate. In Section 2.3.2, we introduce the technicality of the connectivity-aware CNOT synthesis algorithm PermRowCol [26]. Here, we propose a noise-aware CNOT circuit routing algorithm by adapting PermRowCol and utilizing the Cost evaluation. It reduces noise-aware CNOT circuit routing to a Steiner tree problem while accounting for nearest-neighbour interactions and CNOT gate error rates. Since its Cost-instructed heuristics make PermRowCol aware of noises, it is named "NAPermRowCol". "NA" stands for "Noise-Aware".

In Section 4.1, we offer an overview of NAPermRowCol, by presenting a comprehensive summary of its workflow and the intuition behind its technical aspects. In Section 4.2, we explain NAPerRowCol's noise-aware adaptation of PermRowCol. To keep things straightforward, we assume a naive qubit mapping strategy that assigns logical qubit i to physical qubit i. It is important to mention that NAPermRowCol is designed to be compatible with an arbitrary initial qubit mapping strategy.

4.1 The Workflow of NAPermRowCol

NAPermRowCol synthesizes $\mathbf{A}$ by carrying out a sequence of reduction steps. Before each reduction step, a pivot row r and a pivot column c are selected based on the connectivity and edge weights of G, the Cost evaluation, and the binary structure of $\mathbf{A}$. A reduction step involves two actions: a **column reduction** which transforms column c to a basis vector e_r, and a **row reduction** which transforms row r to a transposed basis vector $e_c^\top$. Both reductions are achieved by applying a sequence of Steiner-tree-instructed row operations on $\mathbf{A}$. After a reduction step, vertex r is removed from G. The algorithm terminates when there is one vertex left in G. In the meantime, $\mathbf{A}$ is reduced to a permutation matrix $\mathbf{P}$. Since each row operation corresponds to a CNOT gate, each reduction step outputs a CNOT sequence. NAPermRowCol concatenates these CNOT gates and returns a synthesized circuit $\mathbf{C}_{syn}$ composed of allowed CNOT operations, with $\mathbf{P}$ for qubit relabeling. More precisely, $\mathbf{C}_{syn}$ is semantically equivalent to $\mathbf{C}$ up to permuting logical qubits in quantum registers.

Since SWAP gates are factored out of $\mathbf{C}_{syn}$, the synthesized CNOT count is drastically eliminated. Since noise-aware greedy heuristics are applied to build and traverse a weighted Steiner tree, the cheapest solution is picked for each reduction step and the reliability of a synthesized CNOT subcircuit is maximized. As a result, NAPermRowCol produces a NISQ-executable CNOT circuit for $\mathbf{C}$ with reduced circuit execution time and enhanced fidelity. In what follows, we break down the crux of NAPermRowCol through a two-step explanation.

First, we explain how to find the cheapest sequence of row operations for a reduction step. Given a pivot column c, let S_0 be the set of rows that have a parity of 1. $r \in S_0$ and $|S_0| > 1$. Build a Steiner tree T_0 of G where r is the root and S_0 is the terminal. The Steiner nodes correspond to the rows that have a parity of 0. Procedure 4.2 finds the cheapest path to move the parity of terminal nodes to Steiner nodes. After this traversal, all Steiner nodes will carry a parity of 1. Then traverse T_0 from the leaves to the root and add every parent p to its child c. After the second traversal, the parity 1 from the root will be propagated to every other node in T_0. As a result, every row in column c has a parity of 0 except for row r. Column c is reduced to the basis vector e_r and the column reduction is completed.

Given a pivot row r, we start by solving a system of linear equations and find rows in $\mathbf{A}$ such that $\bigoplus r_k = e_c^\top \oplus r$. Let S_1 be the set of these indices k including the pivot index r. Build a Steiner tree T_1 where r is the root and S_1 is the terminal. Detailed in Procedure 4.3, the first traversal of T_1 is similar to Procedure 4.2, except that u and v have their roles exchanged. Next, traverse T_1 from the leaves to the root and add every child c to its parent p. After the second traversal, the parity on each terminal node is propagated to the root and added together. Since the Steiner nodes are added twice modulo 2 throughout the two traversals, they do not participate in the desired parity summation. As a result, every column in row r has a parity of 0 except for column c. Row r is reduced to the basis vector $e_c^\top$ and the row reduction is completed.

Second, we explain the noise-aware pivot selection. Since each row corresponds to a physical qubit, the removal of vertex r must not disconnect G. Among all non-cut vertices, Procedure 4.4 prioritizes rows with the lowest Hamming weight, followed by the rows tied to vertices having the lowest average incident edge weight. After selecting the pivot row, Procedure 4.5 exhaustively calculates the minimum Cost of each candidate column according to Procedure 4.2, and then picks the one that induces the cheapest solution as the pivot column.

Finally, we consolidate all components and outline the complete workflow of NAPermRowCol.

Procedure 4.1. *To synthesize a CNOT circuit $\mathbf{C}$ according to a NISQ architecture, let $\mathbf{A}$ be $\mathbf{C}$'s parity matrix. $G = (V_G, E_G, \omega_G)$ is an undirected edge-weighted connected graph characterizing the physical restrictions. $\omega_G : E_G \to \{x \in \mathbb{R};\ 0 \le x < 1\}$. For $e = (u, v) \in E_G$, $\omega_G(e)$ is the error rate of coupling physical qubits u and v. NAPermRowCol takes $\mathbf{A}$ and G as inputs. When $\mathbf{A}$ is not a permutation matrix, proceed as follows.*

1. *Procedure 4.4 picks a pivot row r.*
2. *Procedure 4.5 picks a pivot column c.*
3. *Procedure 4.2 carries out a column reduction on $\mathbf{A}$. Let $\mathbf{A}^0$ be the transformed parity matrix.*
4. *Procedure 4.3 carries out a row reduction on $\mathbf{A}^0$. Let $\mathbf{A}^1$ be the transformed parity matrix.*
5. *Remove vertex r from G.*

6. *Remove the pivot row and column from* $\boldsymbol{A}^1$. *Let* $\boldsymbol{A}$ *be the updated parity matrix.*

7. *Go to step 1 until there is precisely one vertex left in* G.

8. *Assemble the permutation matrix* $\boldsymbol{P}$ *based on the reduced row and column in each reduction step.*

9. *Concatenate row operations output from each reduction step. Lemma 2.3 returns a synthesized circuit* $\boldsymbol{C}_{syn}$ *with* $\boldsymbol{P}$.

Thanks to the implementation of the Cost function, NAPermRowCol is scalable and has the potential to route a more complicated quantum circuit on NISQ hardware. Compared to algorithm GENNS whose reduction step halts due to an invalid row operation [74], it is not restricted to an initial qubit map. Compared to the Qiskit transpiler, it does not require ancillary qubits, making it more efficient in terms of resource usage and more suitable for large-scale noise-aware circuit routing.

4.2 Noise-Aware Heuristics

Similar to PermRowCol, NAPermRowCol proceeds by iteratively selecting a pivot row and column, then reducing them to basis vectors with a sequence of row operations. Given a pivot column c, let S_0 be the set of rows that have a parity of 1. In the nontrivial case, $|S_0| > 1$. Let $T_0 = \mathsf{Steiner}(G, S_0)$. A traversal of T_0 is represented by an ordered set of edges, w_{T_0}. When $|w_{T_0}| = t$, it corresponds to applying t row operations on $\mathbf{A}$. Let p_i be the weight of the i-th edge in w_{T_0}, $i \in \mathbb{Z}_t$. The Cost of w_{T_0} is calculated as

$$\mathsf{Cost}(w_{T_0}) = 1 - \prod_{i=0}^{m-1}(1 - \alpha p_i), \qquad \alpha = 1 + \frac{2^{n-2}-1}{2^n+1}. \tag{25}$$

Recall that there are two traversals in Procedure 2.3. In NAPermRowCol, we optimize the first traversal based on the Cost function, keeping the second traversal the same as in PermRowCol.

Procedure 4.2. *In* $T_0 = \mathsf{Steiner}(G, S_0)$, *to find the cheapest path to move the parity of terminal nodes to Steiner nodes, proceed as follows.*

1. *For a Steiner node* $v \in V_{T_0} \setminus S_0$ *that has not yet been picked: (1) For every terminal node* $u \in S_0$, *use Dijkstra's Algorithm to find the cheapest path from* u *to* v *according to the* Cost *evaluation in Equation (25). (2) Pick the cheapest one among them. Express it as an ordered set and add it to the collection of optimal solutions* Opt.

2. *Go to step 1 until all Steiner nodes are considered.*

3. *Use the union operation for all ordered sets in* Opt *to find a combined solution.*

4. *Return the ordered set of the combined solution with its* Cost *evaluation.*

For every candidate column, the traversal on T_0 and its associated Cost are saved in memory. This enables us to quickly access the Cost evaluation and corresponding row operations after a pivot column is selected. As a result, this enhances the time efficiency of NAPermRowCol.

Lemma 4.1. *Given* $T_0 = \mathsf{Steiner}(G, S_0)$, *Procedure 4.2 is well-defined. In other words, it is not possible to encounter a situation depicted in Figure 17, where the cheapest paths for different Steiner nodes cross over each other. By "cross over", we mean two paths share an edge* (a, b). *One path goes from* a *to* b, *while the other goes from* b *to* a.

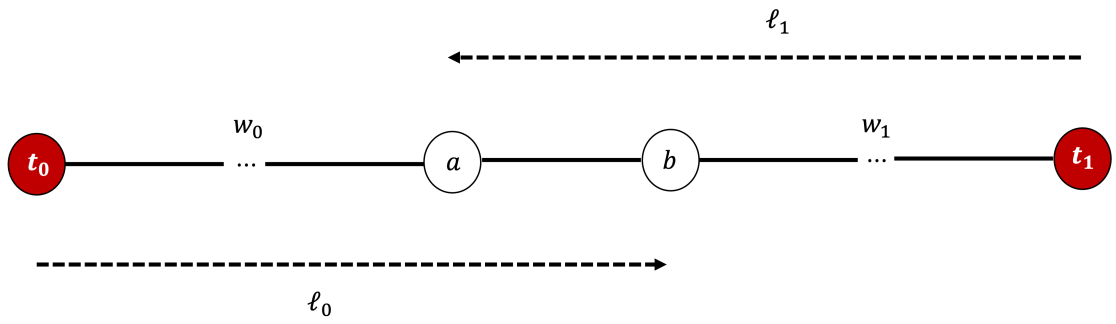


Figure 17: In $T_0 = \mathsf{Steiner}(G, S_0)$, a and b are Steiner nodes. (a,b) denotes the edge between them with weight q. Among all terminal nodes, let t_0 and t_1 be the ones that have the cheapest paths ℓ_0 and ℓ_1 to b and a respectively. Starting from t_0, w_0 is the collection of edges on ℓ_0 between t_0 and a. Starting from t_1, w_1 is the collection of edges on ℓ_1 between t_1 and b.

Proof. Suppose towards contradiction that the cheapest path ℓ_1 for a and the cheapest path ℓ_0 for b overlaps on edge (a,b). According to Procedure 4.2,

$$\mathsf{Cost}(\ell_0) \leq \mathsf{Cost}(w_1), \qquad \mathsf{Cost}(\ell_1) \leq \mathsf{Cost}(w_0). \tag{26}$$

Suppose there are m_0 and m_1 edges in w_0 and w_1 respectively. q is the edge weight of (a,b). p_i and p_j denote the weight of the i-th edge in w_0 and the j-th edge in w_1. $0 \leq i < m_0$, $0 \leq j < m_1$. Let α be the constant specified in Definition 3.3. Then

$$\mathsf{Cost}(w_0) = 1 - \prod_{i=0}^{m_0-1}(1-\alpha p_i), \qquad \mathsf{Cost}(w_1) = 1 - \prod_{j=0}^{m_1-1}(1-\alpha p_j). \tag{27}$$

$$\mathsf{Cost}(\ell_0) = 1 - \left(\prod_{i=0}^{m_0-1}(1-\alpha p_i)\right)(1-\alpha q), \qquad \mathsf{Cost}(\ell_1) = 1 - \left(\prod_{j=0}^{m_1-1}(1-\alpha p_j)\right)(1-\alpha q). \tag{28}$$

Combining Equations (26) to (28), we have

$$1 - \left(\prod_{i=0}^{m_0-1}(1-\alpha p_i)\right)(1-\alpha q) \leq 1 - \prod_{j=0}^{m_1-1}(1-\alpha p_j). \tag{29}$$

$$1 - \left(\prod_{j=0}^{m_1-1}(1-\alpha p_j)\right)(1-\alpha q) \leq 1 - \prod_{i=0}^{m_0-1}(1-\alpha p_i). \tag{30}$$

From Equations (29) and (30), we have

$$\prod_{j=0}^{m_1-1}(1-\alpha p_j) \leq \left(\prod_{i=0}^{m_0-1}(1-\alpha p_i)\right)(1-\alpha q) = \prod_{i=0}^{m_0-1}(1-\alpha p_i) - \alpha q\left(\prod_{i=0}^{m_0-1}(1-\alpha p_i)\right). \tag{31}$$

$$\prod_{i=0}^{m_0-1}(1-\alpha p_i) \leq \left(\prod_{j=0}^{m_1-1}(1-\alpha p_j)\right)(1-\alpha q) = \prod_{j=0}^{m_1-1}(1-\alpha p_j) - \alpha q\left(\prod_{j=0}^{m_1-1}(1-\alpha p_j)\right). \tag{32}$$

According to the empirical data of IBM's fake backends, the error rate of any CNOT gate is bounded by 0.1. By Remark 3.2, $0 < \alpha p_i, \alpha p_j, \alpha q < 1$. From Equation (31),

$$0 < \alpha q\left(\prod_{i=0}^{m_0-1}(1-\alpha p_i)\right) \leq \prod_{i=0}^{m_0-1}(1-\alpha p_i) - \prod_{j=0}^{m_1-1}(1-\alpha p_j). \tag{33}$$

From Equation (32),

$$\prod_{i=0}^{m_0-1}(1-\alpha p_i) - \prod_{j=0}^{m_1-1}(1-\alpha p_j) \leq -\alpha q\left(\prod_{j=0}^{m_1-1}(1-\alpha p_j)\right) < 0. \tag{34}$$

Equations (33) and (34) yields a contradiction. Hence, there is no crossover between the cheapest paths of different Steiner nodes. □

Recall that there are two traversals in Procedure 2.4. In NAPermRowCol, we optimize the first traversal based on the Cost function, keeping the second traversal the same as in PermRowCol.

Procedure 4.3. *In $T_1 = \mathsf{Steiner}(G, S_1)$, to find the cheapest path to move the parity of Steiner nodes to terminal nodes, proceed as follows.*

1. *For a Steiner node $v \in V_{T_1} \setminus S_1$ that has not yet been picked: (1) For every terminal node $u \in S_1$, use Dijkstra's Algorithm to find the cheapest path from v to u according to the* Cost *evaluation in Equation (25). (2) Pick the cheapest one among them. Express it as an ordered set and add it to the collection of optimal solutions* Opt.
2. *Go to step 1 until all Steiner nodes are considered.*
3. *Use the union operation for all ordered sets in* Opt *to find a combined solution.*
4. *Return the ordered set of the combined solution with its* Cost *evaluation.*

Lemma 4.2. *Given $T_1 = \mathsf{Steiner}(G, S_1)$, Procedure 4.3 is well-defined. In other words, it is not possible to encounter a situation depicted in Figure 18, where the cheapest paths for different Steiner nodes cross over each other. By "cross over", we mean two paths share an edge (a, b). One path goes from a to b, while the other goes from b to a.*

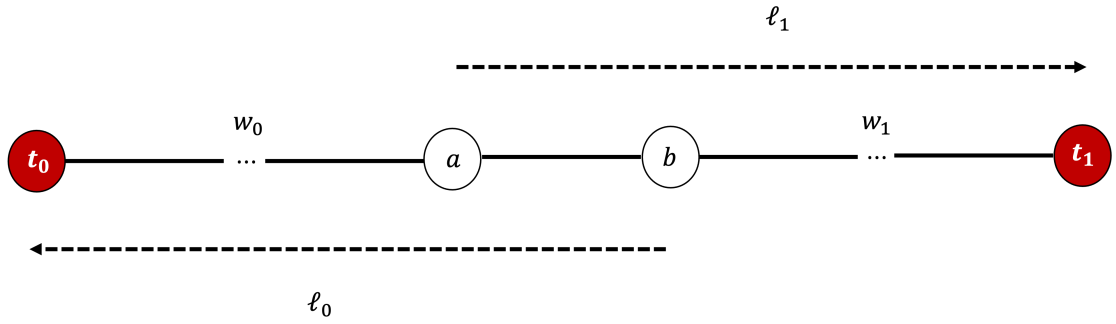


Figure 18: In $T_1 = \mathsf{Steiner}(G, S_1)$, a and b are Steiner nodes. (a, b) denotes the edge between them with weight q. Among all terminal nodes, let t_0 and t_1 be the ones that have the cheapest paths ℓ_0 and ℓ_1 from b and a respectively. Ending at t_0, w_0 is the collection of edges on ℓ_0 between t_0 and a. Ending at t_1, w_1 is the collection of edges on ℓ_1 between t_1 and b.

The symmetry of the graph traversal in Lemma 4.1 leads to the validity of Lemma 4.2. Next, we adjust Procedure 2.1 with a greedy heuristic. Since each row corresponds to a physical qubit, the removal of vertex r must not disconnect G.

Procedure 4.4. *To select a pivot row from $\mathbf{A}$, proceed as follows.*

1. *Among all non-cut vertices, select rows with the lowest Hamming weight to form R_0.*
2. *From R_0, pick rows tied to vertices having the lowest average incident edge weight to form R_1.*
3. *Choose any row from R_1 as the pivot, noting its index as r.*

Lastly, we adjust Procedure 2.2 by leveraging the noise-aware heuristic in Procedure 4.2.

Procedure 4.5. *Given the pivot row r, to select a pivot column from $\mathbf{A}$, proceed as follows.*

1. *Among all columns in $\mathbf{A}$, find the set of columns that have a non-zero entry at row r. Let it be C_0.*
2. *If there exists a basis vector in C_0, let it be the pivot column and note its index as c.*
3. *Otherwise, for each column $c \in C_0$: (1) Build a Steiner tree $T_0 = \mathsf{Steiner}(G, S_0)$ with r as its root. S_0 is the set of rows that have a parity of 1 in c. $|S_0| > 1$. (2) Input T_0 to Procedure 4.2.*
4. *From C_0, select columns with the cheapest* Cost *to form C_1.*
5. *Choose any column from C_1 as the pivot, noting its index as c.*

5 Benchmark Results

IBM's fake backends mimic the behaviours of its quantum computers using system snapshots. They contain important information such as coupling maps, basic gates, qubit qualities (e.g., T1 and T2 time), and gate error rates. In our simulation of a noisy CNOT circuit, we consider the coupling map and use three backends of different sizes: fake Nairobi (7 qubits, Figure 23), fake Guadalupe (16 qubits, Figure 24), and fake Cairo (27 qubits, Figure 25). They are one of the most well-developed IBM's backends with relatively low CNOT gate error rates. Moreover, they have distinct topologies, which help us evaluate the adaptability of NAPermRowCol to different backends. In addition, they are commonly used for benchmarking different synthesis algorithms in the recent literature, so we use them to carry out in-depth comparisons between different CNOT synthesis algorithms. In this section, we discuss the simulation and benchmark results on the fake Nairobi backend with randomly generated CNOT circuits. In Appendices D and E, we report the results using the fake Guadalupe and Cairo backends.

When a circuit consists of n qubits, it is of **width** n. The input dataset[1] consists of CNOT circuits of widths 5, 7, and 16. Using the same circuit generation method as [26, 39], we extend this dataset with circuits of widths 6 and 7. To create more variations of input circuits, we increase the CNOT count for circuits of the same width. For versions of important Python packages and our system specifications, please check out Appendix B.

In Section 5.1, for a synthesized CNOT circuit $\mathbf{C}_{syn}$, we compare the cost function $\mathsf{Cost}(\mathbf{C}_{syn})$ (Definition 3.3) with other commonly used cost functions against the error probability $\mathsf{Prob}(\mathbf{C}_{syn}) = 1 - F_{avg}(\mathbf{C}_{syn})$. Due to the limited scalability of calculating $F_{avg}(\mathbf{C}_{syn})$, our comparison is restricted to $\mathbf{C}_{syn}$ of width no more than 7. Based on the simulation results, $\mathsf{Cost}(\mathbf{C}_{syn})$ fits $\mathsf{Prob}(\mathbf{C}_{syn})$ up to 10^{-3}. In addition, its simple expression allows us to efficiently calculate the cost associated with a noisy CNOT circuit. To synthesize a large CNOT circuit, we use Cost to measure a circuit's noise levels and instruct the error mitigation strategy.

In Section 5.2, we compare the performance of different synthesis algorithms in terms of the Cost metric and the **synthesized CNOT count** (i.e., the gate count of a synthesized CNOT circuit that is physically executable on NISQ hardware). NAPermRowCol performs significantly better than Qiskit transpiler, especially for circuits with a larger number of CNOT gates. Compared with algorithms that are noise-agnostic (PermRowCol and ROWCOL), NAPermRowCol lowers the synthesized CNOT count and is much cheaper for synthesizing random CNOT circuits of different sizes (i.e., different widths and the original CNOT counts). According to the benchmark results, NAPermRowCol provides an improved solution to the CNOT circuit synthesis problem. It not only reduces the CNOT counts, but also minimizes the overall error probabilities.

5.1 Compare Different Cost Functions

Utilizing the Qiskit Python package, we compute the average gate fidelity of a noisy CNOT circuit via superoperator simulation. Since this requires a substantial amount of resources, we calculate $F_{avg}(\mathbf{C}_{syn})$ where $\mathbf{C}_{syn}$ has no more than 7 qubits. Next, compare $\mathsf{Cost}(\mathbf{C}_{syn})$ with two other cost functions to see how well they fit $\mathsf{Prob}(\mathbf{C}_{syn})$ respectively. $\mathsf{Cost1}(\mathbf{C}_{syn})$ is the cost function used in [74]. $\mathsf{Cost2}(\mathbf{C}_{syn})$ is an alternative one based on the probability theory, i.e., the failure rate of $\mathbf{C}_{syn}$. p_i is the error rate of the i-th noisy CNOT gate in $\mathbf{C}_{syn}$ and n is the circuit width.

$$\mathsf{Cost}(\mathbf{C}_{syn}) = 1 - \prod_i (1 - \alpha p_i), \qquad \alpha = 1 + \frac{2^{n-2} - 1}{2^n + 1}.$$
$$\mathsf{Cost1}(\mathbf{C}_{syn}) = \sum_i p_i.$$
$$\mathsf{Cost2}(\mathbf{C}_{syn}) = 1 - \prod_i (1 - p_i).$$

Simulate noisy CNOT circuits with a small gate count To compare different cost functions against $\mathsf{Prob}(\mathbf{C}_{syn})$, consider CNOT circuits of width $n \in \{5,6,7\}$, with m original CNOT count. $m = 2^k$, $2 \le k \le 7$, $k \in \mathbb{N}$. For

[1] https://github.com/Aerylia/pyzx/tree/rowcol/circuits/steiner

each combination of n and m, generate 100 random circuits. According to the topology in Figure 23, synthesize them with NAPermRowCol. Denote the synthesized circuit as $\mathbf{C}_{syn}$ and regroup them based on their CNOT count. For circuits of the same synthesized CNOT count, calculate $\mathsf{Prob}(\mathbf{C}_{syn})$, $\mathsf{Cost}(\mathbf{C}_{syn})$, $\mathsf{Cost1}(\mathbf{C}_{syn})$, and $\mathsf{Cost2}(\mathbf{C}_{syn})$ for each one of them and take the average. By "small gate count", we consider the synthesized CNOT count bounded by 40. For brevity, we drop $\mathbf{C}_{syn}$ from each expression in the remainder of this section. Besides, we use Prob, Cost, Cost1, and Cost2 to denote the value after taking the average for all synthesized circuits of the same CNOT count.

In Figure 19, the comparison is conducted using two methods. In the left column, Root Mean Square Error (RMSE) is used to measure the average difference between the error probability and each cost function. For Cost, Cost1, and Cost2, their RMSEs with respect to Prob correspond to the vertical error bars in green, blue, and orange. Suppose there are N synthesized CNOT circuits with a fixed gate count. For $i \in \mathbb{Z}_N$, let $\mathbf{C}_i$ be each such circuit. The length of the green error bar is calculated as

$$\sqrt{\frac{1}{N}\sum_{i=0}^{N-1}\big(\mathsf{Prob}(\mathbf{C}_i)-\mathsf{Cost}(\mathbf{C}_i)\big)^2}. \tag{35}$$

RMSE serves as a goodness-of-fit assessment and evaluates how accurately each cost function approximates Prob. It ranges between 0 and positive infinity. As the cost function moves closer to Prob, the approximation has less error, and thus has better precision. A value of 0 (almost impossible in practice) indicates a perfect fit to Prob. RMSE is a simple metric that provides a straightforward interpretation of a cost function's overall error [6].

For circuits of widths 5, 6, and 7 with different CNOT counts, Cost demonstrates a minor deviation from Prob. Compared with Cost1 and Cost2, it approximates Prob with higher precision. As shown in the left columns of Figures 26 and 27, on the other IBM's backends, Cost consistently provides a much more accurate approximation for Prob than the other cost functions. These simulation results show that Cost is an accountable approximation for Prob, despite varied topology and error distribution across different backends.

In the right column of Figure 19, the maximum distance between Prob and each cost function is used as an alternative metric for the goodness-of-fit assessment. Suppose there are N synthesized CNOT circuits with gate count M. For $i \in \mathbb{Z}_N$, let $\mathbf{C}_i$ be each such circuit. The maximum distance between Prob and Cost is defined as

$$d(\mathsf{Prob},\mathsf{Cost}) = \max_{i\in\mathbb{Z}_N}|\mathsf{Prob}(\mathbf{C}_i)-\mathsf{Cost}(\mathbf{C}_i)|. \tag{36}$$

Since $d(\mathsf{Prob},\mathsf{Cost})$ has a wide range, we use a logarithmic scale for the y-axis to plot all data compactly. For circuits of widths $n \in \{5,6,7\}$, Cost fits Prob up to 10^{-3}, much closer than Cost1 and Cost2. Similarly, in the right columns of Figures 26 and 27, Cost is a tighter approximation for Prob than Cost1 and Cost2. It fits Prob up to 10^{-1} for all synthesized CNOT counts.

There is another interesting pattern between different cost functions. In the right column of Figure 19, when the synthesized CNOT count increases, $d(\mathsf{Prob},\mathsf{Cost})$ increases, so does $d(\mathsf{Prob},\mathsf{Cost1})$. However, $d(\mathsf{Prob},\mathsf{Cost2})$ increases when the synthesized CNOT count is lower than 20. Then it decreases when the synthesized CNOT counts get larger. This is because when the CNOT count is low, the accumulated CNOT error rate is also low. As the CNOT count increases, the growth of Cost2 surpasses the growth of Prob, so their distance is reduced. To see if Cost2 will outperform Cost, we use synthesized CNOT circuits with higher CNOT counts to investigate whether the distance between Prob and Cost2 will eventually decrease to 0.

Simulate noisy CNOT circuits with a larger gate count To randomly generate synthesized CNOT circuits, our previous method has its limitations. The synthesized CNOT count does not scale well with the number of input circuits to NAPermRowCol. For example, if we use circuits with 128 original CNOT count as input, no synthesized circuit has a CNOT count of more than 44. To this end, we use a topology-based method to generate a synthesized CNOT circuit with a larger gate count. By "larger gate count", we consider the synthesized circuit whose gate count is bounded by 70.

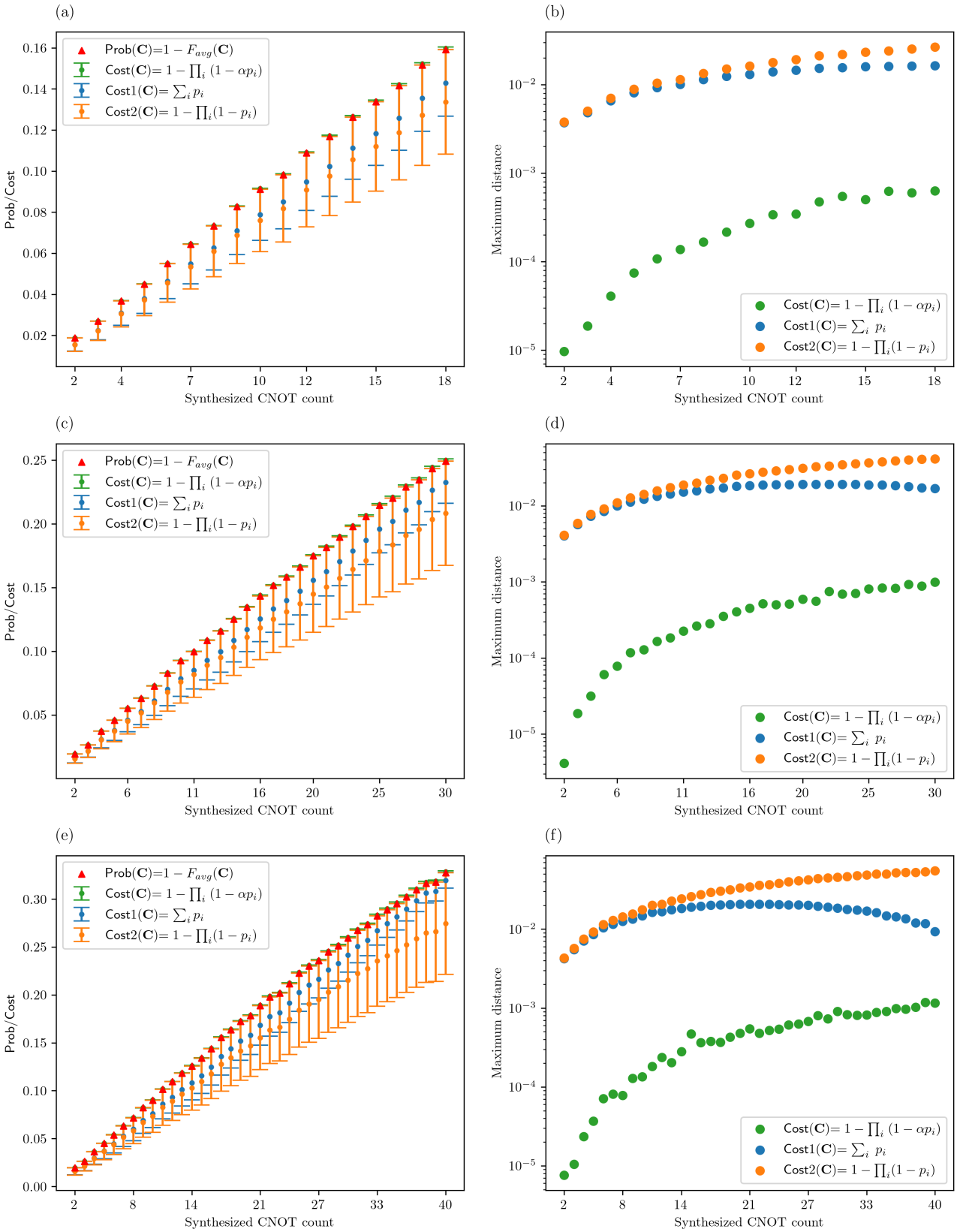
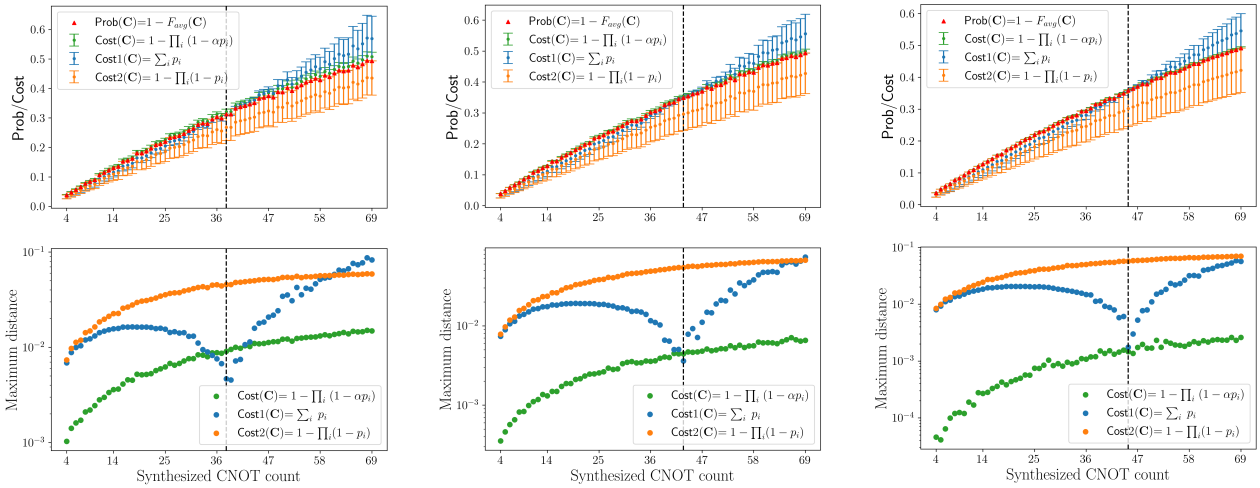


Figure 19: On IBM's fake Nairobi backend, compare the three cost functions against the error probability of a synthesized CNOT circuit. p_i is the error rate of a noisy CNOT gate, n is the circuit width, $\alpha = 1 + \frac{2^{n-2}-1}{2^n+1}$. In the left column, figures (a), (c), and (e) report results about the synthesized CNOT circuits of widths 5, 6, and 7 respectively. In each figure, for circuits of the same synthesized CNOT count, a y-value corresponds to the average of the error probability or the average of a cost function. The length of an error bar measures the average difference between the error probability and each cost function. The longer error bar, the worse the approximation. In the right column, we compare the maximum distance between the error probability and each cost function. Figures (b), (d), and (f) report results about the synthesized CNOT circuits of widths 5, 6, and 7 respectively. Since $d(\mathsf{Prob}, \mathsf{Cost})$ has a wide range, we use a logarithmic scale for the y-axis to see all the numbers easily.



(a) Synthesized CNOT circuits of width 5. (b) Synthesized CNOT circuits of width 6. (c) Synthesized CNOT circuits of width 7.

Figure 20: On IBM's fake Nairobi backend, compare the three cost functions against the error probability of a synthesized CNOT circuit with gate count ranging between 4 and 69. p_i is the error rate of a noisy CNOT gate, n is the circuit width, $\alpha = 1 + \frac{2^{n-2}-1}{2^n+1}$. In each figure of the first row, for circuits of the same synthesized CNOT count, a y-value corresponds to the average of the error probability or the average of a cost function. The length of an error bar measures the average difference between the error probability and each cost function. The longer error bar, the worse the approximation. In the second row, we compare the maximum distance between the error probability and each cost function. Since $d(\mathsf{Prob}, \mathsf{Cost})$ has a wide range, we use a logarithmic scale for the y-axis to see all the numbers easily. In each column, the black dashed line corresponds to a synthesized CNOT count of 38, 43, and 45. This denotes the point where $d(\mathsf{Prob}, \mathsf{Cost1})$ attains the minimum value, which nearly coincides with $d(\mathsf{Prob}, \mathsf{Cost})$. The extended input dataset enhances our confidence in the goodness of Cost as an approximation for Prob.

Let G be a hardware topology, $G = (V_G, E_G)$, $|V_G| = N$ and $|E_G| = M$. Compose a synthesized CNOT circuit of width n with m gates by considering only the allowed CNOT operations based on G, $n \leq N$. That is, the CNOTs whose control and target correspond to two adjacent vertices in G. Since a CNOT is asymmetric with its control and target, each edge $e = (u, v) \in E_G$ has an orientation. For a directed edge, let its head and tail be the control and target of a CNOT respectively.

1. Let $\mathsf{total} = 0$, $\mathbf{C} = \emptyset$, E'_G is the set of 2M directed edges of E_G.
2. Randomly pick one directed edge $e \in E'_G$.
3. While $\mathsf{total} < m$:
   (1) Right-append $\mathbf{C}$ by e.
   (2) $\mathsf{total} = \mathsf{total} + 1$.
   (3) $e = e'$.
   (4) Randomly pick one directed edge $e \in E'_G \setminus \{e'\}$.
4. Return $\mathbf{C}$.

Figure 20 compare the goodness-of-fit of each cost function for the error probability of larger synthesized CNOT circuits. In the first row, for circuits of the same synthesized CNOT count, each error bar measures the average difference between the error probability and each cost function. For all circuit widths, Cost1 intersects with Prob, before which the difference between Prob and Cost1 decreases. This means Cost1 provides a better fit for Prob when the synthesized CNOT count goes beyond a certain point. Such a changing point is annotated by the black dashed line in each column. It corresponds to a synthesized CNOT count of 38, 43, and 45. After the intersection, the difference between Prob and Cost1 increases, indicating a worse fit for Prob when the synthesized CNOT count increases. In comparison, the difference between Prob and Cost remains small for all synthesized CNOT count.

In the second row, we compare the maximum distance between the error probability and every cost function. In each column, the black dashed line from the first row denotes the point where $d(\mathsf{Prob}, \mathsf{Cost1})$ attains the minimum value (between 10^{-3} and 10^{-2}), which nearly coincides with $d(\mathsf{Prob}, \mathsf{Cost})$. This implies that for certain synthesized CNOT count, Cost1 provides a good approximation for Prob. However, in general, Cost gives a more accurate approximation for circuits of different synthesized CNOT counts.

To summarize, on different IBM's backends and for CNOT circuits of different sizes, Cost fits Prob significantly better than the other cost functions. Moreover, it circumvents the computation complexity of calculating the average gate fidelity and shows remarkable scalability. Therefore, we use it to design the noise-aware CNOT synthesis algorithm NAPermRowCol and evaluate the performance of our error mitigation strategy.

5.2 Compare Different CNOT Synthesis Algorithms

NAPermRowCol is inspired by the algorithm GENNS [74] and built upon the algorithm PermRowCol [26]. It reduces a noise-aware CNOT routing algorithm to a Steiner tree problem and lowers the CNOT count by factoring out SWAP gates after CNOT synthesis. Moreover, it uses noise-aware heuristics to make an informed decision at each reduction step. To gauge its performance, we benchmark it against the state-of-the-art CNOT synthesis algorithms according to the Cost metric and the synthesized CNOT count. Our comparative analysis is carried out between NAPermRowCol, PermRowCol, ROWCOL [71], and Qiskit. Qiskit is short for "Qiskit transpilation at optimization level 3". It implements the SWAP-based heuristic algorithm SABRE [41]. These algorithms take a logical CNOT circuit $\mathbf{C}$ and an undirected edge-weighted connected graph $G = (V_G, E_G, \omega_G)$ as inputs and return a synthesized circuit with allowed CNOT operations. Although it would be ideal to include GENNS in our comparison, it fails upon an invalid initial qubit map. Consider an arbitrary logical qubit map to a connected subgraph of G. Since every other algorithm works with such a map, the incompatibility of GENNS makes it unsuitable for a fair comparison.

In this section, we analyze the benchmark results using IBM's fake Nairobi backend. In Appendices E.2 and E.3, we report results on IBM's fake Guadalupe and Cairo backends. For the Nairobi backend, the input dataset consists of CNOT circuits of width $n \in \{5,7\}$. For Guadalupe and Cairo backends, the input dataset is extended with CNOT circuits of width 16. Let m be the original CNOT count of the input circuit. For all backends and circuit widths, $m = 2^k$, $2 \leq k \leq 10$. For each combination (n,m), randomly generate 100 circuits. All other experimental conditions remain the same across different backends.

To find a good initial qubit map for each original CNOT circuit $\mathbf{C}$, we start by considering the one selected by Qiskit. It randomly generates a map for all logical qubits and synthesizes $\mathbf{C}$ with SWAP gates. After repeating this several times, it finds the best map Φ and returns the synthesized circuit. If Φ maps logical qubits to connected quantum registers, we use it as the initial qubit map for ROWCOL, PermRowCol, and NAPermRowCol. Otherwise, these algorithms fail as they assume a connected topology. Alternatively, select a subset of connected vertices, $V_{G'} \subset V_G$, $|V_{G'}| = n$. Let $G' = (V_{G'}, E_{G'}, \omega_{G'})$ be the induced subgraph of G. $\omega_{G'}(e) = \omega_G(e)$, $e \in E_{G'}$. Assign each logical qubit to a vertex in $V_{G'}$ in increasing order. That is, if $i < j$, $\Phi(i) < \Phi(j)$. For example, to map a 5-qubit logical circuit to a connected subgraph of Figure 23, let $V_{G'} = \{0,1,2,3,4\}$. Then the naive map Φ is described by Table 1.

Logical qubit i	0	1	2	3	4
Vertex j	0	1	2	3	4

Table 1: For clarity, we use a green label to denote a logical qubit and a blue label to denote a physical qubit. The naive qubit map can be expressed as $\Phi = [0,1,2,3,4]$.

After applying Φ to the logical qubits of $\mathbf{C}$, input it alongside G' into the benchmarked algorithms. For each combination (n,m), let $\mathbf{C}_i$ be the synthesized circuits returned by NAPermRowCol, $0 \leq i \leq 99$. Find the synthesized gate count of $\mathbf{C}_i$ and $\mathsf{Cost}(\mathbf{C}_i)$, then take the average for both of them. Repeat the same procedure for algorithms PermRowCol and ROWCOL. Since Qiskit handles the initial qubit mapping and returns a synthesized circuit $\mathbf{C}_i$ based on G, for each combination (n,m), we can also average its synthesized

gate count and $\mathsf{Cost}(\mathbf{C}_i)$, for $0 \leq i \leq 99$. To optimize the benchmark's runtime, we use the Python package Ray [42] to parallelize the execution of these four synthesis algorithms.

In Appendix C, Tables 4 and 5 demonstrate benchmark results of synthesizing 5- and 7-qubit CNOT circuits on IBM's fake Nairobi backend. According to the Cost metric and the synthesized CNOT count, NAPermRowCol outperforms PermRowCol, ROWCOL, and Qiskit for all input circuits of different CNOT counts. This is also supported by the benchmark results on IBM's other backends. For more details, we encourage readers to check out Tables 6 and 7 and Figures 28 and 29 for CNOT synthesis on the fake Guadalupe backend, as well as Tables 8 and 9 and Figures 31 and 32 for CNOT synthesis on the fake Cairo backend. For the Nairobi backend, we discuss these benchmark results from two perspectives.

Compare the scalability of different synthesis algorithms Let m be the original CNOT count. In Figures 21a and 21c, when m is small, the performance of each algorithm is barely distinguishable. When m increases, the synthesized CNOT count and the Cost of each algorithm increase. Among them, Qiskit shows the worst scalability. In Figure 21a,

- When $m \leq 8$, for each algorithm, its synthesized CNOT count is bounded by 12, and its Cost is bounded by 0.12.
- When $m \geq 16$, Qiskit's synthesized CNOT count grows exponentially, while those of NAPermRowCol, PermRowCol, and ROWCOL remain nearly unchanged. They are bounded by 13, 14, and 20 respectively.
- When $m \geq 256$, the synthesized CNOT count of these three algorithms is about 100 times fewer than that of Qiskit.

Similarly, in Figure 21c,

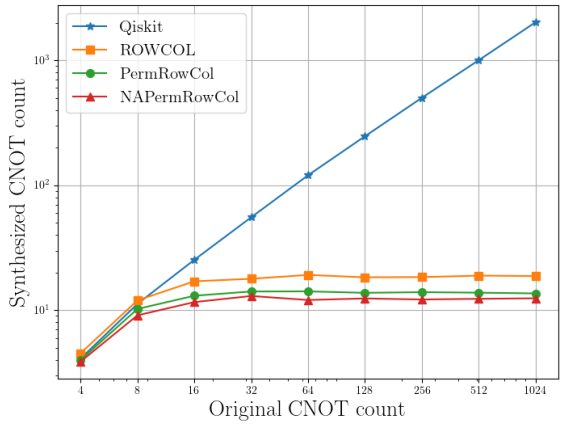
- When $m \geq 16$, Qiskit's Cost explodes, while those of NAPermRowCol, PermRowCol, ROWCOL fluctuate around 0.11, 0.13, and 0.17 respectively.
- When $m \geq 256$, Qiskit's Cost saturates the maximum possible cost and reaches 1. In comparison, the Cost of NAPermRowCol, PermRowCol, and ROWCOL remains quite low. It is bounded by 0.12, 0.14, and 0.18 respectively, which is more than 5 times cheaper than Qiskit.

In summary, NAPermRowCol, PermRowCol, and ROWCOL show impressive and comparable scalability when synthesizing a large 5-qubit CNOT circuit.

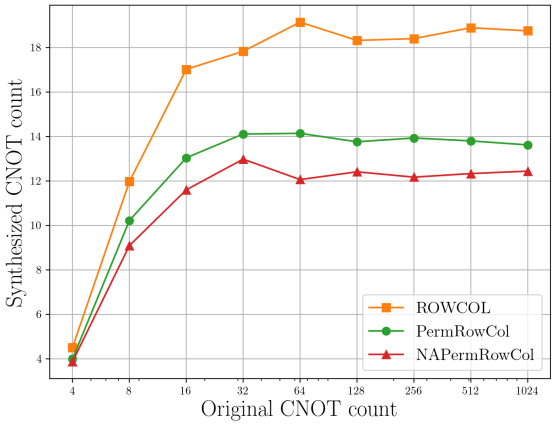
Compare the relative performance of NAPermRowCol, PermRowCol, and ROWCOL Compared to Figures 21a and 21c, Figures 21b and 21d get rid of the data related to Qiskit so that the remaining ones are distributed in a more compact area. They serve as the zoomed-in versions which allow us to compare the performance of NAPermRowCol, PermRowCol, and ROWCOL more closely. Compared to PermRowCol, NAPermRowCol reduces the synthesized CNOT count by about 16%, and it is about 13% cheaper. Compared to ROWCOL, it reduces the synthesized CNOT count by about 42%, and it is about 48% cheaper. In summary, both the Cost metric and the synthesized CNOT count show that NAPermRowCol outperforms PermRowCol and ROWCOL.

These results also confirm the effectiveness of the noise-aware heuristics introduced in Section 4.2. In a reduction step, NAPermRowCol may take a detour to avoid expensive edges. Under both metrics, NAPermRowCol consistently beats PermRowCol. This means our greedy approach of selecting the cheapest path at each reduction step results in an additional improvement to optimize the overall synthesis results.

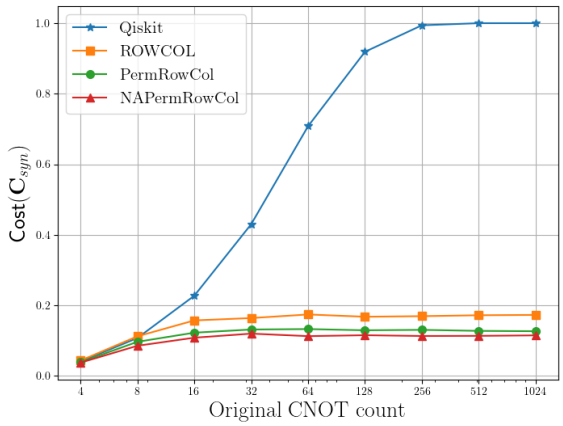
Figure 22 demonstrate benchmark results of synthesizing a 7-qubit CNOT circuit on IBM's fake Nairobi backend. As in Figure 21, Qiskit shows the worst scalability when the original CNOT count increases. Under both metrics, NAPermRowCol outperforms PermRowCol, ROWCOL, and Qiskit for all input circuits of different CNOT counts. In addition, it shows that when synthesizing a wider CNOT circuit, the performance of each algorithm declines. Compare to Figure 21b, in Figure 22b, the maximum synthesized CNOT count of each algorithm has increased. For example, the synthesized CNOT count of NAPermRowCol is bounded by



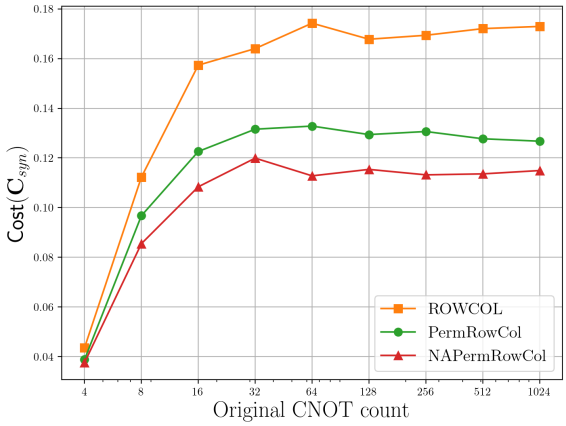
(a) Compare NAPermRowCol with PermRowCol, ROWCOL, and Qiskit in terms of the synthesized CNOT count. Qiskit has the worst scalability when the original CNOT count grows exponentially.



(b) Compare NAPermRowCol with PermRowCol and ROWCOL in terms of the synthesized CNOT count. This is the zoomed-in version of the lefthand side.



(c) Compare NAPermRowCol with PermRowCol, ROWCOL, and Qiskit in terms of their respective costs. Qiskit has the worst scalability when the original CNOT count grows exponentially.



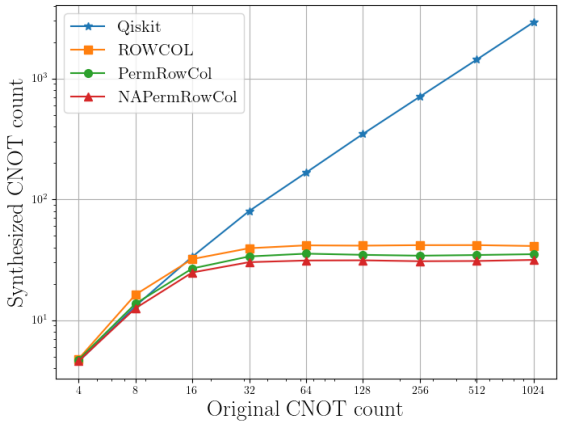
(d) Compare NAPermRowCol with PermRowCol and ROWCOL in terms of their respective costs. This is the zoomed-in version of the lefthand side.

Figure 21: IBM's fake Nairobi backend hosts 7 qubits. We benchmark with its 5-qubit connected subgraph and compare NAPermRowCol against three state-of-the-art CNOT synthesis algorithms. For each original CNOT count, input 100 randomly generated CNOT circuits to each algorithm, obtain the synthesized CNOT circuits, then average their gate count and the circuit cost. The x-axis in each figure uses a logarithmic scale as the input gate count grows exponentially. The y-axis of Figure 21a uses a logarithmic scale, while the ones in Figures 21b to 21d use a linear scale. Compared to Figures 21a and 21c, Figures 21b and 21d get rid of the data related to Qiskit so that the remaining ones are distributed in a more compact area. They serve as the zoomed-in versions which allow us to compare the performance of NAPermRowCol, PermRowCol, and ROWCOL more closely. For all input circuits of different CNOT counts, NAPermRowCol outperforms other algorithms in terms of the synthesized CNOT count and circuit cost. It demonstrates remarkable scalability when the input circuit size grows exponentially.

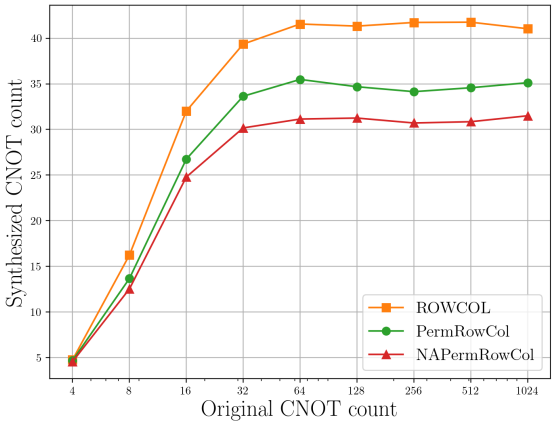
31, but it is bounded by 12 when synthesizing a 5-qubit CNOT circuit. Similarly, in Figure 22d, the Cost of NAPermRowCol is bounded by 0.27, but it is bounded by 0.115 in Figure 21d.

Moreover, the advantages of NAPermRowCol over PermRowCol and ROWCOL are less obvious. Compared to PermRowCol, NAPermRowCol reduces the synthesized CNOT count by about 13%, and it is about 11% cheaper. Compared to ROWCOL, it reduces the synthesized CNOT count by about 35.5%, and it is about 26% cheaper. In summary, the improvement of NAPermRowCol is suppressed when synthesizing a wider CNOT circuit. We can draw the same conclusion based on the benchmark results on IBM's other backends. When the CNOT circuit has more than 15 qubits, the performance between NAPermRowCol, PermRowCol, and ROWCOL is barely distinguishable. For more details, we encourage readers to check out Figures 30 and 33 for synthesizing 16-qubit CNOT circuits on the fake Guadalupe and Cairo backends.

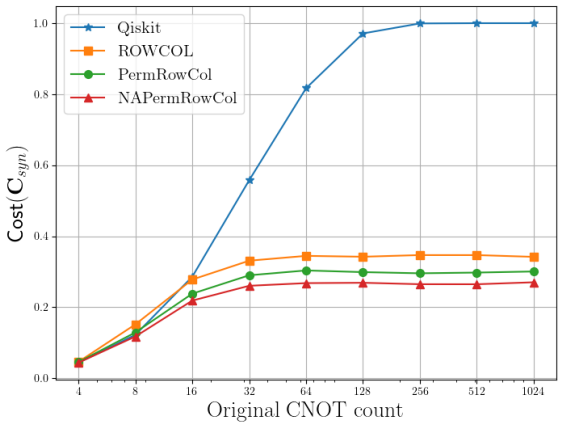
Finally, we compare the performance of NAPermRowCol with Qiskit, whose routing strategy uses ancillary qubits to connect subsets of non-adjacent qubits. In contrast, NAPermRowCol and other algorithms do not use ancilla and thus have much better scalability. In terms of the algorithm performance, NAPermRowCol is not as versatile as Qiskit, with a less-than-optimal strategy at each reduction step. Despite such an unfair comparison, NAPermRowCol consistently performs much better than Qiskit on all benchmarked backends, especially for CNOT circuits of large gate counts. This highlights the potential of NAPermRowCol and our error mitigation strategy to route a more complicated quantum circuits on various NISQ hardware.



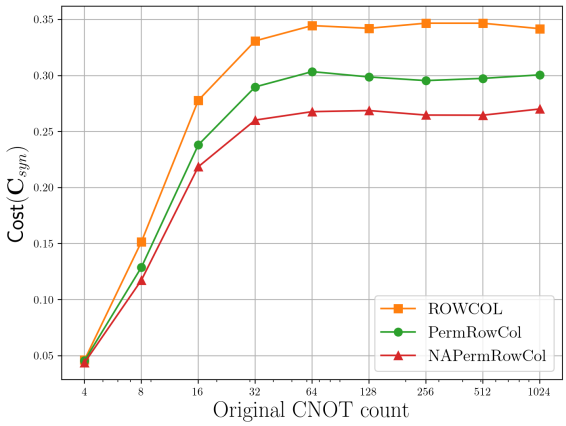
(a) Compare NAPermRowCol with PermRowCol, ROWCOL, and Qiskit in terms of the synthesized CNOT count. Qiskit has the worst scalability when the original CNOT count grows exponentially.



(b) Compare NAPermRowCol with PermRowCol and ROWCOL in terms of the synthesized CNOT count. This is the zoomed-in version of the lefthand side.



(c) Compare NAPermRowCol with PermRowCol, ROWCOL, and Qiskit in terms of their respective costs. Qiskit has the worst scalability when the original CNOT count grows exponentially.



(d) Compare NAPermRowCol with PermRowCol and ROWCOL in terms of their respective costs. This is the zoomed-in version of the lefthand side.

Figure 22: IBM's fake Nairobi backend hosts 7 qubits. We benchmark with its 7-qubit connected subgraph and compare NAPermRowCol against three state-of-the-art CNOT synthesis algorithms. For each original CNOT count, input 100 randomly generated CNOT circuits to each algorithm, obtain the synthesized CNOT circuits, then average their gate count and the circuit cost. The x-axis in each figure uses a logarithmic scale as the input gate count grows exponentially. The y-axis of Figure 22a uses a logarithmic scale, while the ones in Figures 22b to 22d use a linear scale. Compared to Figures 22a and 22c, Figures 22b and 22d get rid of the data related to Qiskit so that the remaining ones are distributed in a more compact area. They serve as the zoomed-in versions which allow us to compare the performance of NAPermRowCol, PermRowCol, and ROWCOL more closely. For input circuits of more than 16 CNOT counts, NAPermRowCol outperforms other algorithms in terms of the synthesized CNOT count and circuit cost. It demonstrates remarkable scalability when the input circuit size grows exponentially.

6 Acknowledgement

The diagrams were typeset using TikZit [40]. The authors would like to thank Christopher Wilson for consultations about superconducting quantum computers; Kohdai Kuroiwa for suggestions to improve the Cost function; Arianne Meijer - van de Griend and Ewan Murphy for discussing PermRowCol and its benchmark details; Gyungmin Cho and Chanung Park for fruitful discussions on approximating the average gate fidelity; Neil J. Ross, Peter Selinger, John van de Wetering, as well as anonymous reviewers at the Theory of Quantum Computation, Communication and Cryptography (TQC) and Quantum Physics and Logic (QPL) for helpful comments on an earlier version of this paper.

MK and DK gratefully acknowledge the support of a National Research Foundation of Korea (NRF) grant funded by the Korean Government (MSIT) (No. 2019M3E4A1080144, No. 2019M3E4A1080145, No. 2019R1A5A1027055, RS-2023-00283291, SRC Center for Quantum Coherence in Condensed Matter RS-2023-00207732, and No. 2023R1A2C2005809) and a core center program grant funded by the Ministry of Education (No. 2021R1A6C101B418). SML and MM thank NTT Research for their financial and technical support. This work was supported in part by Canada's NSERC. Research at IQC is supported in part by the Government of Canada through Innovation, Science and Economic Development Canada. Research at Perimeter Institute is supported in part by the Government of Canada through the Department of Innovation, Science and Economic Development Canada and by the Province of Ontario through the Ministry of Colleges and Universities.

References

[1] Google Quantum AI (2023): *Suppressing quantum errors by scaling a surface code logical qubit*. Nature 614(7949), pp. 676–681, doi:https://doi.org/10.1038/s41586-022-05434-1.

[2] Gernot Alber, Thomas Beth, Michał Horodecki, Paweł Horodecki, Ryszard Horodecki, Martin Rötteler, Harald Weinfurter, Reinhard Werner, Anton Zeilinger, Thomas Beth et al. (2001): *Quantum algorithms: Applicable algebra and quantum physics*. Quantum information: an introduction to basic theoretical concepts and experiments, pp. 96–150, doi:https://doi.org/10.1007/3-540-44678-8_4.

[3] Matthew Amy, Parsiad Azimzadeh & Michele Mosca (2018): *On the controlled-NOT complexity of controlled-NOT–phase circuits*. Quantum Science and Technology 4(1), doi:https://doi.org/10.1088/2058-9565/aad8ca.

[4] Erin Angelini & Hugh Collins: *IBM Debuts Next-Generation Quantum Processor & IBM Quantum System Two, Extends Roadmap to Advance Era of Quantum Utility*. https://newsroom.ibm.com/2023-12-04-IBM-Debuts-Next-Generation-Quantum-Processor-IBM-Quantum-System-Two,-Extends-Roadmap-to-Advance-Era-of-Quantum-Utility. Accessed: 2024-03-03.

[5] Erin Angelini & Hugh Collins: *IBM Quantum Computer Demonstrates Next Step Towards Moving Beyond Classical Supercomputing*. https://newsroom.ibm.com/2023-06-14-IBM-Quantum-Computer-Demonstrates-Next-Step-Towards-Moving-Beyond-Classical-Supercomputing. Accessed: 2024-03-03.

[6] J Scott Armstrong (2001): *Evaluating forecasting methods*. Principles of forecasting: A handbook for researchers and practitioners, pp. 443–472. Available at https://link.springer.com/chapter/10.1007/978-0-306-47630-3_20.

[7] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell et al. (2019): *Quantum supremacy using a programmable superconducting processor*. Nature 574(7779), pp. 505–510, doi:https://doi.org/10.1038/s41586-019-1666-5.

[8] Fergus Barratt, James Dborin, Matthias Bal, Vid Stojevic, Frank Pollmann & Andrew G Green (2021): *Parallel quantum simulation of large systems on small NISQ computers*. npj Quantum Information 7(1), p. 79, doi:https://doi.org/10.1038/s41534-021-00420-3.

[9] Tomas Basile & Carlos Pineda (2023): *Quantum simulation of Pauli channels and dynamical maps: algorithm and implementation*. arXiv preprint arXiv:2308.00188, doi:https://doi.org/10.48550/arXiv.2308.00188.

[10] Kishor Bharti, Alba Cervera-Lierta, Thi Ha Kyaw, Tobias Haug, Sumner Alperin-Lea, Abhinav Anand, Matthias Degroote, Hermanni Heimonen, Jakob S Kottmann, Tim Menke et al. (2022): *Noisy intermediate-scale quantum algorithms*. Reviews of Modern Physics 94(1), p. 015004, doi:10.1103/RevModPhys.94.015004.

[11] Dolev Bluvstein, Simon J Evered, Alexandra A Geim, Sophie H Li, Hengyun Zhou, Tom Manovitz, Sepehr Ebadi, Madelyn Cain, Marcin Kalinowski, Dominik Hangleiter et al. (2023): *Logical quantum processor based on reconfigurable atom arrays*. Nature, pp. 1–3, doi:https://doi.org/10.1038/s41586-023-06927-3.

[12] Mark D Bowdrey, Daniel KL Oi, Anthony J Short, Konrad Banaszek & Jonathan A Jones (2002): *Fidelity of single qubit maps*. Physics Letters A 294(5-6), pp. 258–260, doi:https://doi.org/10.1016/S0375-9601%2802%2900069-5.

[13] Sergey Bravyi, Andrew W Cross, Jay M Gambetta, Dmitri Maslov, Patrick Rall & Theodore J Yoder (2023): *High-threshold and low-overhead fault-tolerant quantum memory*. arXiv preprint arXiv:2308.07915, doi:https://doi.org/10.48550/arXiv.2308.07915.

[14] Jarosław Byrka, Fabrizio Grandoni, Thomas Rothvoß & Laura Sanità (2013): *Steiner tree approximation via iterative randomized rounding*. Journal of the ACM (JACM) 60(1).

[15] Zhenyu Cai (2020): *Resource estimation for quantum variational simulations of the Hubbard model*. Physical Review Applied 14(1), p. 014059, doi:10.1103/PhysRevApplied.14.014059.

[16] Xinyu Chen, Mingqiang Zhu, Xueyun Cheng, Pengcheng Zhu & Zhijin Guan (2023): *Nearest neighbor synthesis of CNOT circuits on general quantum architectures*. arXiv preprint arXiv:2310.00592, doi:https://doi.org/10.48550/arXiv.2310.00592.

[17] Giovanni De Micheli, Jie-Hong R Jiang, Robert Rand, Kaitlin Smith & Mathias Soeken (2022): *Advances in quantum computation and quantum technologies: A design automation perspective*. IEEE Journal on Emerging and Selected Topics in Circuits and Systems 12(3), pp. 584–601, doi:10.1109/JETCAS.2022.3205174.

[18] Phoebus J Dhrymes & Phoebus J Dhrymes (2000): *Matrix vectorization*. Mathematics for Econometrics, pp. 117–145.

[19] IBM Quantum Documentation: *Compute resources*. https://quantum.ibm.com/services/resources?tab=systems. Accessed: 2024-04-28.

[20] IBM Quantum Documentation: *ECRGate*. https://docs.quantum.ibm.com/api/qiskit/qiskit.circuit.library.ECRGate. Accessed: 2024-04-28.

[21] Robert W Floyd (1962): *Algorithm 97: shortest path*. Communications of the ACM 5(6), p. 345.

[22] Evan M Fortunato, Lorenza Viola, Jonathan Hodges, Grum Teklemariam & David G Cory (2002): *Implementation of universal control on a decoherence-free qubit*. New Journal of Physics 4(1), p. 5.

[23] Jay M Gambetta, Jerry M Chow & Matthias Steffen (2017): *Building logical qubits in a superconducting quantum computing system*. npj quantum information 3(1), p. 2.

[24] Felix Gemeinhardt, Antonio Garmendia, Manuel Wimmer, Benjamin Weder & Frank Leymann (2023): *Quantum Combinatorial Optimization in the NISQ Era: A Systematic Mapping Study*. ACM Computing Surveys 56(3), pp. 1–36, doi:https://doi.org/10.1145/3620668.

[25] Vlad Gheorghiu, Jiaxin Huang, Sarah Meng Li, Michele Mosca & Priyanka Mukhopadhyay (2022): *Reducing the CNOT count for Clifford+T circuits on NISQ architectures*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, doi:10.1109/TCAD.2022.3213210.

[26] Arianne Meijer-van de Griend & Sarah Meng Li (2023): *Dynamic Qubit Routing with CNOT Circuit Synthesis for Quantum Compilation*. EPTCS 394(arXiv: 2205.00724), pp. 363–399, doi:https://doi.org/10.48550/arXiv.2205.00724.

[27] Arianne Meijer-van de Griend & Ross Duncan (2020): *Architecture-aware synthesis of phase polynomials for NISQ devices*. arXiv preprint. arXiv:2004.06052.

[28] Riddhi Swaroop Gupta, Claire L Edmunds, Alistair R Milne, Cornelius Hempel & Michael J Biercuk (2020): *Adaptive characterization of spatially inhomogeneous fields and errors in qubit registers*. npj Quantum Information 6(1), p. 53. Available at https://www.nature.com/articles/s41534-020-0286-0.

[29] Kathleen E Hamilton, Tyler Kharazi, Titus Morris, Alexander J McCaskey, Ryan S Bennink & Raphael C Pooser (2020): *Scalable quantum processor noise characterization*. In: 2020 IEEE International Conference on Quantum Computing and Engineering (QCE), IEEE, pp. 430–440.

[30] Ethan Hansen, Sanskriti Joshi & Hannah Rarick (2023): *Resource Estimation of Quantum Multiplication Algorithms*. In: 2023 IEEE International Conference on Quantum Computing and Engineering (QCE), 2, IEEE, pp. 199–202, doi:10.1109/QCE57702.2023.10211.

[31] Sahay Harshvardhan, Sanil Jain, James E McClure, Caleb McIrvin & Ngoc Quy Tran (2023): *Simulating Noisy Quantum Circuits for Cryptographic Algorithms*. arXiv preprint arXiv:2306.02111, doi:https://doi.org/10.48550/arXiv.2306.02111.

[32] Pawel Horodecki, Michal Horodecki & Ryszard Horodecki (1998): *General teleportation channel, singlet fraction and quasi-distillation, arXiv e-prints*. arXiv preprint quant-ph/9807091.

[33] Stefan Hougardy, Jannik Silvanus & Jens Vygen (2015): *Dijkstra meets Steiner: a fast exact goal-oriented Steiner tree algorithm*, doi:https://doi.org/10.48550/arXiv.1406.0492. arXiv:1406.0492.

[34] Frank K Hwang & Dana S Richards (1992): *Steiner tree problems*. Networks 22(1).

[35] Yanjun Ji, Sebastian Brandhofer & Ilia Polian (2022): *Calibration-Aware Transpilation for Variational Quantum Optimization*. In: *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pp. 204–214, doi:10.1109/QCE53715.2022.00040.

[36] Petar Jurcevic, David Zajac, Jiri Stehlik, Isaac Lauer & Ryan Mandelbaum: *Pushing quantum performance forward with our highest Quantum Volume yet*. https://research.ibm.com/blog/quantum-volume-256. Accessed: 2024-02-22.

[37] Richard M Karp (1972): *Reducibility among combinatorial problems*. In: Complexity of computer computations, Springer, pp. 85–103.

[38] Julian Kelly: *A Preview of Bristlecone, Google's New Quantum Processor*. https://blog.research.google/2018/03/a-preview-of-bristlecone-googles-new.html. Accessed: 2024-03-03.

[39] Aleks Kissinger & Arianne Meijer van de Griend (2020): *CNOT circuit extraction for topologically-constrained quantum memories*. Quantum Information and Computation 20(7-8). arXiv:1904.00633.

[40] Aleks Kissinger, Alexander Merry, Chris Heunen & K. Johan Paulsson: *TikZiT*. https://tikzit.github.io/index.html. Accessed: 2024-02-22.

[41] Gushu Li, Yufei Ding & Yuan Xie (2019): *Tackling the qubit mapping problem for NISQ-era quantum devices*. In: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 1001–1014. arXiv:1809.02573.

[42] Eric Liang: *Overview of Ray*. Available at https://docs.ray.io/en/latest/ray-overview/index.html.

[43] Easwar Magesan, Robin Blume-Kohout & Joseph Emerson (2011): *Gate fidelity fluctuations and quantum process invariants*. Phys. Rev. A 84, p. 012309, doi:10.1103/PhysRevA.84.012309. Available at https://link.aps.org/doi/10.1103/PhysRevA.84.012309.

[44] Simon Martiel & Timothée Goubault de Brugière (2022): *Architecture aware compilation of quantum circuits via lazy synthesis*. Quantum 6, p. 729.

[45] Dmitri Maslov, Sean M Falconer & Michele Mosca (2008): *Quantum circuit placement*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 27(4), pp. 752–763, doi:10.1109/TCAD.2008.917562.

[46] Prakash Murali, Jonathan M Baker, Ali Javadi-Abhari, Frederic T Chong & Margaret Martonosi (2019): *Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers*. In: *Proceedings of the twenty-fourth international conference on architectural support for programming languages and operating systems*, pp. 1015–1029, doi:10.1145/3297858.3304075.

[47] Ewan Murphy & Aleks Kissinger (2023): *Global Synthesis of CNOT Circuits with Holes*. arXiv preprint arXiv:2308.16496.

[48] Beatrice Nash, Vlad Gheorghiu & Michele Mosca (2020): *Quantum circuit optimizations for NISQ architectures*. Quantum Science and Technology 5(2), p. 025010, doi:10.1088/2058-9565/ab79b1.

[49] Paul Nation, Hanhee Paik, Andrew Cross & Nazario Zaira: *The IBM Quantum heavy hex lattice*. https://www.ibm.com/quantum/blog/heavy-hex-lattice. Accessed: 2024-03-02.

[50] Michael A Nielsen (2002): *A simple formula for the average gate fidelity of a quantum dynamical operation*. Physics Letters A 303(4), pp. 249–252.

[51] Siyuan Niu, Adrien Suau, Gabriel Staffelbach & Aida Todri-Sanial (2020): *A hardware-aware heuristic for the qubit mapping problem in the nisq era*. IEEE Transactions on Quantum Engineering 1, pp. 1–14.

[52] Ketan N Patel, Igor L Markov & John P Hayes (2004): *Optimal synthesis of linear reversible circuits*. Quantum Information & Computation 8(3), doi:10.26421.

[53] Sundar Pichai: *Our progress toward quantum error correction*. https://blog.google/inside-google/message-ceo/our-progress-toward-quantum-error-correction/. Accessed: 2024-03-03.

[54] Intel PR: *Intel's New Chip to Advance Silicon Spin Qubit Research for Quantum Computing*. https://www.intel.com/content/www/us/en/newsroom/news/quantum-computing-chip-to-advance-research.html#gs.5wm57y. Accessed: 2024-03-03.

[55] Intel PR: *Quantum Computing Systems Achieving Quantum Practicality*. https://www.intel.com/content/www/us/en/research/quantum-computing.html. Accessed: 2024-03-03.

[56] John Preskill (2018): *Quantum computing in the NISQ era and beyond*. *Quantum* 2, p. 79, doi:https://doi.org/10.22331/q-2018-08-06-79.

[57] M.H. Protter & C.B.J. Morrey (2012): *Intermediate Calculus*. Undergraduate Texts in Mathematics, Springer New York. Available at https://books.google.co.kr/books?id=3lTmBwAAQBAJ.

[58] Nils Quetschlich, Lukas Burgholzer & Robert Wille (2023): *Compiler optimization for quantum computing using reinforcement learning*. In: *2023 60th ACM/IEEE Design Automation Conference (DAC)*, IEEE, pp. 1–6.

[59] Gabriel Robins & Alexander Zelikovsky (2005): *Tighter bounds for graph Steiner tree approximation*. *SIAM Journal on Discrete Mathematics* 19(1).

[60] Nicholas Rubin & Ryan Babbush: *Developing industrial use cases for physical simulation on future error-corrected quantum computers*. https://blog.research.google/2023/10/developing-industrial-use-cases-for.html. Accessed: 2024-03-03.

[61] Ciaran Ryan-Anderson, Justin G Bohnet, Kenny Lee, Daniel Gresh, Aaron Hankin, JP Gaebler, David Francois, Alexander Chernoguzov, Dominic Lucchetti, Natalie C Brown et al. (2021): *Realization of real-time fault-tolerant quantum error correction*. *Physical Review X* 11(4), p. 041058, doi:10.1103/PhysRevX.11.041058.

[62] Benjamin Schumacher (1996): *Sending entanglement through noisy quantum channels*. *Physical Review A* 54(4), p. 2614, doi:10.1103/PhysRevA.54.2614.

[63] Simone Severini: *Bye NISQ. Hello LISQ?* https://www.linkedin.com/pulse/bye-nisq-hello-lisq-simone-severini-ybkmc/. Accessed: 2024-02-22.

[64] Ali Shaib, Mohamad Hussein Naim, Mohammed E Fouda, Rouwaida Kanj & Fadi Kurdahi (2023): *Efficient noise mitigation technique for quantum computing*. *Scientific Reports* 13(1), p. 3912.

[65] Vivek V Shende, Aditya K Prasad, Igor L Markov & John P Hayes (2003): *Synthesis of reversible logic circuits*. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 22(6), pp. 710–722, doi:https://doi.org/10.48550/arXiv.quant-ph/0207001.

[66] Seyon Sivarajah, Silas Dilkes, Alexander Cowtan, Will Simmons, Alec Edgington & Ross Duncan (2020): *t|ket>: a retargetable compiler for NISQ devices*. *Quantum Science and Technology* 6(1). arXiv:2003.10611.

[67] Neereja Sundaresan, Theodore J Yoder, Youngseok Kim, Muyuan Li, Edward H Chen, Grace Harper, Ted Thorbeck, Andrew W Cross, Antonio D Córcoles & Maika Takita (2023): *Demonstrating multi-round subsystem quantum error correction using matching and maximum likelihood decoders*. *Nature Communications* 14(1), p. 2852, doi:https://doi.org/10.1038/s41467-023-38247-5.

[68] Vivien Vandaele, Simon Martiel & Timothée Goubault de Brugière (2022): *Phase polynomials synthesis algorithms for NISQ architectures and beyond*. *Quantum Science and Technology*. arXiv:2104.00934.

[69] Jean-Baptiste Waring, Christophe Pere & Sebastien Le Beux (2024): *Noise Aware Utility Optimization of NISQ Devices*. *arXiv preprint arXiv:2402.08226*, doi:https://doi.org/10.48550/arXiv.2402.08226.

[70] Robert Wille, Rod Van Meter & Yehuda Naveh (2019): *IBM's Qiskit tool chain: Working with and developing for real quantum computers*. In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, pp. 1234–1240, doi:10.23919/DATE.2019.8715261.

[71] Bujiao Wu, Xiaoyu He, Shuai Yang, Lifu Shou, Guojing Tian, Jialin Zhang & Xiaoming Sun (2023): *Optimization of CNOT circuits on limited-connectivity architecture*. *Physical Review Research* 5(1), p. 013065.

[72] Qian Xu, J Ataides, Christopher A Pattison, Nithin Raveendran, Dolev Bluvstein, Jonathan Wurtz, Bane Vasic, Mikhail D Lukin, Liang Jiang & Hengyun Zhou (2023): *Constant-overhead fault-tolerant quantum computation with reconfigurable atom arrays*. *arXiv preprint arXiv:2308.08648*, doi:https://doi.org/10.48550/arXiv.2308.08648.

[73] Xiangzhen Zhou, Yuan Feng & Sanjiang Li (2022): *Quantum Circuit Transformation: A Monte Carlo Tree Search Framework*. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*. arXiv:2008.09331.

[74] Mingqiang Zhu, Xueyun Cheng, Pengcheng Zhu, Liang Chen & Zhijin Guan (2022): *Physical constraint-aware CNOT quantum circuit synthesis and optimization*. *Quantum Information Processing* 22(1), p. 10, doi:https://doi.org/10.1007/s11128-022-03716-1.

A Complementary Proofs

In this section, we provide complementary proof details for statements in Sections 2.4.2 and 3.

Lemma 2.13.

$$F_{pro}(\mathcal{E}') = \frac{Tr[S_{\mathcal{E}'}]}{t^2}.$$

Proof. According to [22, 62], the process fidelity can be expressed as

$$F_{pro}(\mathcal{E}') = \sum_k |\mathsf{Tr}[U^\dagger M_k]/t|^2. \tag{37}$$

By direct computation, we rewrite Equation (37) as

$$\begin{aligned}
F_{pro}(\mathcal{E}') &= \frac{1}{t^2}\sum_k (\mathsf{Tr}[U^\dagger M_k])^* \mathsf{Tr}[U^\dagger M_k] \\
&= \frac{1}{t^2}\sum_k \mathsf{Tr}[U^T M_k^*]\mathsf{Tr}[U^\dagger M_k] \\
&= \frac{1}{t^2}\sum_k \mathsf{Tr}[(U^T M_k^*)\otimes(U^\dagger M_k)] \\
&= \frac{1}{t^2}\mathsf{Tr}\Big[\sum_k (U^T M_k^*)\otimes(U^\dagger M_k)\Big] \\
&= \frac{1}{t^2}\mathsf{Tr}\Big[\sum_k (U^\dagger M_k)^*\otimes(U^\dagger M_k)\Big] \\
&= \frac{1}{t^2}\mathsf{Tr}\Big[\sum_k M_k'^*\otimes M_k'\Big].
\end{aligned} \tag{38}$$

By Lemma 2.7, since the action of $\mathcal{E}'$ is described by $\{M_k';\ \ M_k' = U^\dagger M_k\}$, $\sum_k (M_k^*\otimes M_k) = S_{\mathcal{E}'}$. Hence,

$$F_{pro}(\mathcal{E}') = \frac{\mathsf{Tr}[S_{\mathcal{E}'}]}{t^2}.$$

□

Lemma 3.1. *Let $\mathcal{E}_q$ be an n-qubit channel that is vertically composed of two noisy channels $\mathcal{E}_0$ and $\mathcal{E}_1$, with input dimension d_0 and d_1, error probability p_0 and p_1. $\mathcal{E}_q = \mathcal{E}_0 \otimes \mathcal{E}_1$, $d_q = d_0 d_1 = 2^n$. d_q is the input dimension of $\mathcal{E}_q$. Then*

$$F_{ave}(\mathcal{E}_q) = 1 - p_0 - p_1 + p_0 p_1 + \frac{(1-d_1)p_0 + (1-d_0)p_1 + (d_0+d_1)p_0 p_1}{d_0 d_1 + 1}.$$

Proof. Applying Lemma 2.11 with Equations (19) and (20), we have

$$\begin{aligned}
F_{avg}(\mathcal{E}_q) = F_{avg}(\mathcal{E}_0 \otimes \mathcal{E}_1) &= \frac{\mathsf{Tr}[S_{\mathcal{E}_0 \otimes \mathcal{E}_1}]}{(d_0 d_1)^2} \times \frac{d_0 d_1}{d_0 d_1 + 1} + \frac{1}{d_0 d_1 + 1} \\
&= \frac{\mathsf{Tr}[S_{\mathcal{E}_0}]}{d_0^2} \frac{\mathsf{Tr}[S_{\mathcal{E}_1}]}{d_1^2} \times \frac{d_0 d_1}{d_0 d_1 + 1} + \frac{1}{d_0 d_1 + 1} \\
&= \left(1 - \frac{d_0 + 1}{d_0} p_0\right)\left(1 - \frac{d_1 + 1}{d_1} p_1\right) \frac{d_0 d_1}{d_0 d_1 + 1} + \frac{1}{d_0 d_1 + 1} \\
&= 1 - \left(\frac{d_0 + 1}{d_0} p_0 + \frac{d_1 + 1}{d_1} p_1\right) \frac{d_0 d_1}{d_0 d_1 + 1} + \frac{(d_0 + 1)(d_1 + 1)}{d_0 d_1 + 1} p_0 p_1 \\
&= 1 - \left(\frac{d_0 d_1 + d_1 + 1 - 1}{d_0 d_1 + 1} p_0 + \frac{d_0 d_1 + d_0 + 1 - 1}{d_0 d_1 + 1} p_1\right) + \frac{d_0 d_1 + d_0 + d_1 + 1}{d_0 d_1 + 1} p_0 p_1 \\
&= 1 - p_0 - \frac{d_1 - 1}{d_0 d_1 + 1} p_0 - p_1 - \frac{d_0 - 1}{d_0 d_1 + 1} p_1 + p_0 p_1 + \frac{(d_0 + d_1) p_0 p_1}{d_0 d_1 + 1} \\
&= 1 - p_0 - p_1 + p_0 p_1 + \frac{(1 - d_1) p_0 + (1 - d_0) p_1 + (d_0 + d_1) p_0 p_1}{d_0 d_1 + 1}
\end{aligned}$$

□

Lemma 3.4. *Let $t = 2^n$ be the input dimension of $\mathcal{E}_c$. For $k \in \mathbb{Z}_2$, $d_k = t$ is the input dimension of an error channel $\mathcal{E}_k$ with error probability p_k. If $\mathcal{E}_c = \mathcal{E}_1 \circ \mathcal{E}_0$ and $A_k = I - S_{\mathcal{E}_k}$,*

$$(t+1)^2 p_0 p_1 \leq Tr[A_1 A_0] \leq 2(t+1)^2 p_0 p_1.$$

Proof. According to Corollary 2.1, for $k \in \mathbb{Z}_2$,

$$A_k = I - S_{\mathcal{E}_k} = I - \sum_{E_i \in \mathcal{B}_n} P_i^k S_{E_i}^k. \tag{39}$$

Let E_0^k be the $t^2 \times t^2$ identity matrix I with probability P_0^k. Equation (39) can be expressed as

$$A_k = I - P_0^k E_0^k - \sum_{E_i \in \mathcal{B}_n \setminus \{E_0^k\}} P_i^k S_{E_i} = (1 - P_0^k) I - \sum_{E_i \in \mathcal{B}_n \setminus \{I\}} P_i^k S_{E_i}. \tag{40}$$

It follows that

$$\begin{aligned}
A_1 A_0 &= \left((1 - P_0^1) I - \sum_{E_i \in \mathcal{B}_n \setminus \{I\}} P_i^1 S_{E_i}\right)\left((1 - P_0^0) I - \sum_{E_j \in \mathcal{B}_n \setminus \{I\}} P_j^0 S_{E_j}\right) \\
&= (1 - P_0^1)(1 - P_0^0) I - (1 - P_0^1) \sum_{E_j \in \mathcal{B}_n \setminus \{I\}} P_j^0 S_{E_j} - (1 - P_0^0) \sum_{E_i \in \mathcal{B}_n \setminus \{I\}} P_i^1 S_{E_i} + \sum_{\substack{E_i \in \mathcal{B}_n \setminus \{I\} \\ E_j \in \mathcal{B}_n \setminus \{I\}}} (P_i^1 S_{E_i})(P_j^0 S_{E_j}).
\end{aligned} \tag{41}$$

By linearity and Lemma 2.9,

$$\begin{aligned}
\mathsf{Tr}[A_1A_0] &= \mathsf{Tr}\Bigg[\big(1-P_0^1\big)\big(1-P_0^0\big)I - \big(1-P_0^1\big)\sum_{E_j\in\mathcal{B}_n\setminus\{I\}} P_j^0 S_{E_j} - \big(1-P_0^0\big)\sum_{E_i\in\mathcal{B}_n\setminus\{I\}} P_i^1 S_{E_i} \\
&\qquad + \sum_{\substack{E_i\in\mathcal{B}_n\setminus\{I\}\\ E_j\in\mathcal{B}_n\setminus\{I\}}} \big(P_i^1 S_{E_i}\big)\big(P_j^0 S_{E_j}\big)\Bigg] \\
&= \mathsf{Tr}\Big[\big(1-P_0^1\big)\big(1-P_0^0\big)I\Big] - \mathsf{Tr}\Big[\big(1-P_0^1\big)\sum_{E_j\in\mathcal{B}_n\setminus\{I\}} P_j^0 S_{E_j}\Big] - \mathsf{Tr}\Big[\big(1-P_0^0\big)\sum_{E_i\in\mathcal{B}_n\setminus\{I\}} P_i^1 S_{E_i}\Big] \\
&\quad + \mathsf{Tr}\Bigg[\sum_{\substack{E_i\in\mathcal{B}_n\setminus\{I\}\\ E_j\in\mathcal{B}_n\setminus\{I\}}} \big(P_i^1 S_{E_i}\big)\big(P_j^0 S_{E_j}\big)\Bigg]. \\
&= \big(1-P_0^1\big)\big(1-P_0^0\big)t^2 - \big(1-P_0^1\big)\sum_{E_j\in\mathcal{B}_n\setminus\{I\}} P_j^0\,\mathsf{Tr}[S_{E_j}] - \big(1-P_0^0\big)\sum_{E_i\in\mathcal{B}_n\setminus\{I\}} P_i^1\,\mathsf{Tr}[S_{E_i}] \\
&\qquad + \sum_{\substack{E_i\in\mathcal{B}_n\setminus\{I\}\\ E_j\in\mathcal{B}_n\setminus\{I\}}} P_i^1 P_j^0\,\mathsf{Tr}[S_{E_i}S_{E_j}]. \\
&= \big(1-P_0^1\big)\big(1-P_0^0\big)t^2 + \sum_{\substack{E_i\in\mathcal{B}_n\setminus\{I\}\\ E_j\in\mathcal{B}_n\setminus\{I\}}} P_i^1 P_j^0\,\mathsf{Tr}[S_{E_i}S_{E_j}]. \qquad (42)
\end{aligned}$$

For $1 \le i, j \le 4^n - 1$, since E_i and E_j are unitary, by Lemma 2.6,

$$S_{E_i}S_{E_j} = (E_i^* \otimes E_i)(E_j^* \otimes E_j) = (E_i^*E_j^*) \otimes (E_iE_j). \qquad (43)$$

Since

$$\mathsf{Tr}[(E_i^*E_j^*) \otimes (E_iE_j)] = \mathsf{Tr}[E_i^*E_j^*]\mathsf{Tr}[E_iE_j] = \big(\mathsf{Tr}[E_jE_i]\big)^*\mathsf{Tr}[E_iE_j],$$

combined with Equations (42) and (43) and lemma 2.8, we have

$$\begin{aligned}
\mathsf{Tr}[A_1A_0] &= \big(1-P_0^1\big)\big(1-P_0^0\big)t^2 + \sum_{\substack{E_i\in\mathcal{B}_n\setminus\{I\}\\ E_j\in\mathcal{B}_n\setminus\{I\}}} P_i^1 P_j^0 \big(\mathsf{Tr}[E_jE_i]\big)^*\mathsf{Tr}[E_iE_j]. \\
&= \big(1-P_0^1\big)\big(1-P_0^0\big)t^2 + \sum_{i=1}^{4^n-1} P_i^1 P_i^0 t^2. \\
&= t^2\Big(\big(1-P_0^1\big)\big(1-P_0^0\big) + \sum_{i=1}^{4^n-1} P_i^1 P_i^0\Big). \qquad (44)
\end{aligned}$$

By Equation (19) and lemma 2.9,

$$t^2\left(1 - \frac{t+1}{t}p_k\right) = \mathsf{Tr}\Big[\sum_{E_i\in\mathcal{B}_n} P_i^k S_{E_i^k}\Big] = P_0^k\mathsf{Tr}[S_I] + \sum_{i=1}^{4^n-1} P_i^k\mathsf{Tr}[S_{E_i^k}] = t^2 P_0^k.$$

It follows that

$$P_0^k = 1 - \frac{t+1}{t}p_k, \qquad \sum_{i=1}^{4^n-1} P_i^k = 1 - P_0^k = \frac{t+1}{t}p_k. \qquad (45)$$

Define a function $f(P_i^0, P_i^1)$ with domain D as follows.

$$f(P_i^0, P_i^1) = \sum_{i=1}^{4^n-1} P_i^1 P_i^0, \qquad D = \left\{P_i^k;\ \sum_{i=1}^{4^n-1} P_i^k = \frac{t+1}{t}p_k,\ \ k\in\mathbb{Z}_2,\ \ P_i^k \ge 0,\ \ 1 \le i \le 4^n-1\right\}.$$

By Equation (45), Equation (44) can be simplified as

$$\mathrm{Tr}[A_1A_0] = t^2\left(\frac{(t+1)^2}{t^2}p_0p_1 + \sum_{i=1}^{4^n-1} P_i^1P_i^0\right) = (t+1)^2p_0p_1 + t^2 f(P_i^0, P_i^1). \tag{46}$$

To bound Equation (46), our problem is reduced to identifying the maximum and minimum values of f, $f_{\max}$ and $f_{\min}$, on D. By the extreme value theorem, $f_{\max}$ and $f_{\min}$ are either the extremum points of f or located on the boundary of D. Using the Lagrange Multiplier method [57], the extremum point is reached when all P_i^k have the same values. That is, for all $1 \le i \le 4^n - 1$

$$P_i^0 = \frac{(t+1)p_0}{t(t^2-1)}, \qquad P_i^1 = \frac{(t+1)p_1}{t(t^2-1)}.$$

Since $t^2 = 4^n$,

$$f_{ex} = \sum_{i=1}^{4^n-1}\left(\frac{(t+1)p_0}{t(t^2-1)}\right)\left(\frac{(t+1)p_1}{t(t^2-1)}\right) = \frac{(t+1)^2}{(t^2-1)t^2}p_0p_1. \tag{47}$$

For $k \in \mathbb{Z}_2$, $0 \le P_i^k \le \frac{t+1}{t}p_k$. The boundary of D is reached when for all $1 \le i \le t^2-1$, $P_i^1P_i^0 = 0$. Thus $f = 0$. Or for some $1 \le i \le t^2-1$, $P_i^k = \frac{t+1}{t}p_k$. Thus

$$f = \sum_{i=1}^{t^2-1} P_i^1P_i^0 = \left(\frac{t+1}{t}p_0\right)\left(\frac{t+1}{t}p_1\right) + 0 = \frac{(t+1)^2}{t^2}p_0p_1$$

Therefore,

$$f_{\min} = 0, \qquad f_{\max} = \frac{(t+1)^2}{t^2}p_0p_1. \tag{48}$$

Combining Equations (46) and (48), we have

$$(t+1)^2p_0p_1 \le \mathrm{Tr}[A_1A_0] \le 2(t+1)^2p_0p_1.$$

□

B Package Version and Hardware Specifics

To generate simulation and benchmark results, we utilize two distinct hardware environments and various Python packages. The important ones are documented here. For other packages, please refer to the GitHub repository[2]. Qiskit Aer is installed from the source code[3].

Python packages	**Version**
Qiskit	0.45.2
Qiskit Aer	0.13.2
pyzx	0.7.3
System information	**Hardware Specifics**
Python	3.8.17
Operating System	Ubuntu 22.04.2 LTS
CPU	AMD EPYC 7763 64-Core Processor @ 2.45GHz
Memory	1.23 TB

Table 2: The simulation results reported in Section 5.1 and appendix D are generated under these specifications.

[2] https://github.com/Minyoung-Kim1110/NoiseAwareSynthesis
[3] https://qiskit.org/ecosystem/aer/getting_started.html

Python packages	**Version**
Qiskit	0.44.3
Qiskit Terra	0.25.3
Qiskit Aer	0.13.0
System information	**Hardware Specifics**
Python	3.8.18
Operating System	Window 10 Education
CPU	AMD Ryzen Threadripper 3970X 32-core @3.69 GHz
Memory	256 GB

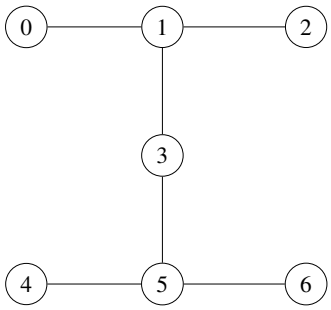
Table 3: The benchmark results reported in Section 5.2 and appendix E are generated under these specifications.

Note that here, "pyzx" is not a package. It is a module from the "rowcol" branch in the GitHub repo[4].

C Backends for Simulation and Benchmarking

We use the empirical data from IBM's fake Nairobi, Guadalupe, and Cairo backends to compare different cost functions and the performance of different CNOT synthesis algorithms.

C.1 IBM's Fake Nairobi Backend



(a) Each vertex corresponds to a physical qubit. Each edge represents a CNOT gate that can be performed on the qubits corresponding to its endpoints.

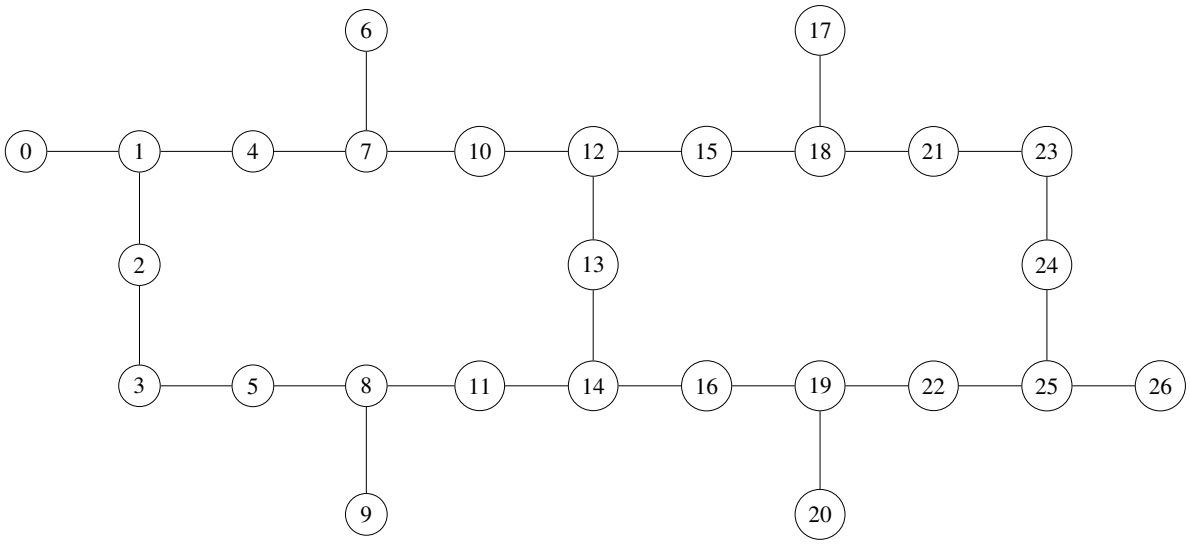
Edge	Edge Weight
(0, 1)	0.00777
(1, 2)	0.00607
(1, 3)	0.00792
(3, 5)	0.01016
(4, 5)	0.00619
(5, 6)	0.00918

(b) For $e = (u, v) \in E_G$, $\omega_G(e)$ is the error rate of coupling physical qubits u and v.

Figure 23: $G = (V_G, E_G, \omega_G)$ is the connectivity graph of IBM's fake Nairobi backend. It is an undirected edge-weighted connected graph. $|V_G| = 7$, $\omega_G : E_G \to \{x \in \mathbb{R};\ 0 \leq x < 1\}$.

C.2 IBM's Fake Guadalupe Backend

C.3 IBM's Fake Cairo Backend



(a) Each vertex corresponds to a physical qubit. Each edge represents a CNOT gate that can be performed on the qubits corresponding to its endpoints.

Edge	Edge Weight	Edge	Edge Weight
(0, 1)	0.025728	(12, 15)	0.008980
(1, 2)	0.006662	(13, 14)	0.004904
(1, 4)	0.011427	(14, 16)	0.005036
(2, 3)	0.011890	(15, 18)	0.005864
(3, 5)	0.005375	(16, 19)	0.007042
(4, 7)	0.016432	(17, 18)	0.009230
(5, 8)	0.004620	(18, 21)	0.005924
(6, 7)	0.014319	(19, 20)	0.007014
(7, 10)	0.022012	(19, 22)	0.005040
(8, 9)	0.006167	(21, 23)	0.008903
(8, 11)	0.053568	(22, 25)	0.023629
(10, 12)	0.006628	(23, 24)	0.003967
(11, 14)	0.013671	(24, 25)	0.023295
(12, 13)	0.014007	(25, 26)	0.028715

(b) For $e = (u, v) \in E_G$, $\omega_G(e)$ is the error rate of coupling physical qubits u and v.

Figure 25: $G = (V_G, E_G, \omega_G)$ is the connectivity graph of IBM's fake Cairo backend. It is an undirected edge-weighted connected graph. $|V_G| = 16$, $\omega_G : E_G \to \{x \in \mathbb{R};\ 0 \leq x < 1\}$.

[4] https://github.com/Aerylia/pyzx

D Compare Different Cost Functions on IBM's Fake Backends

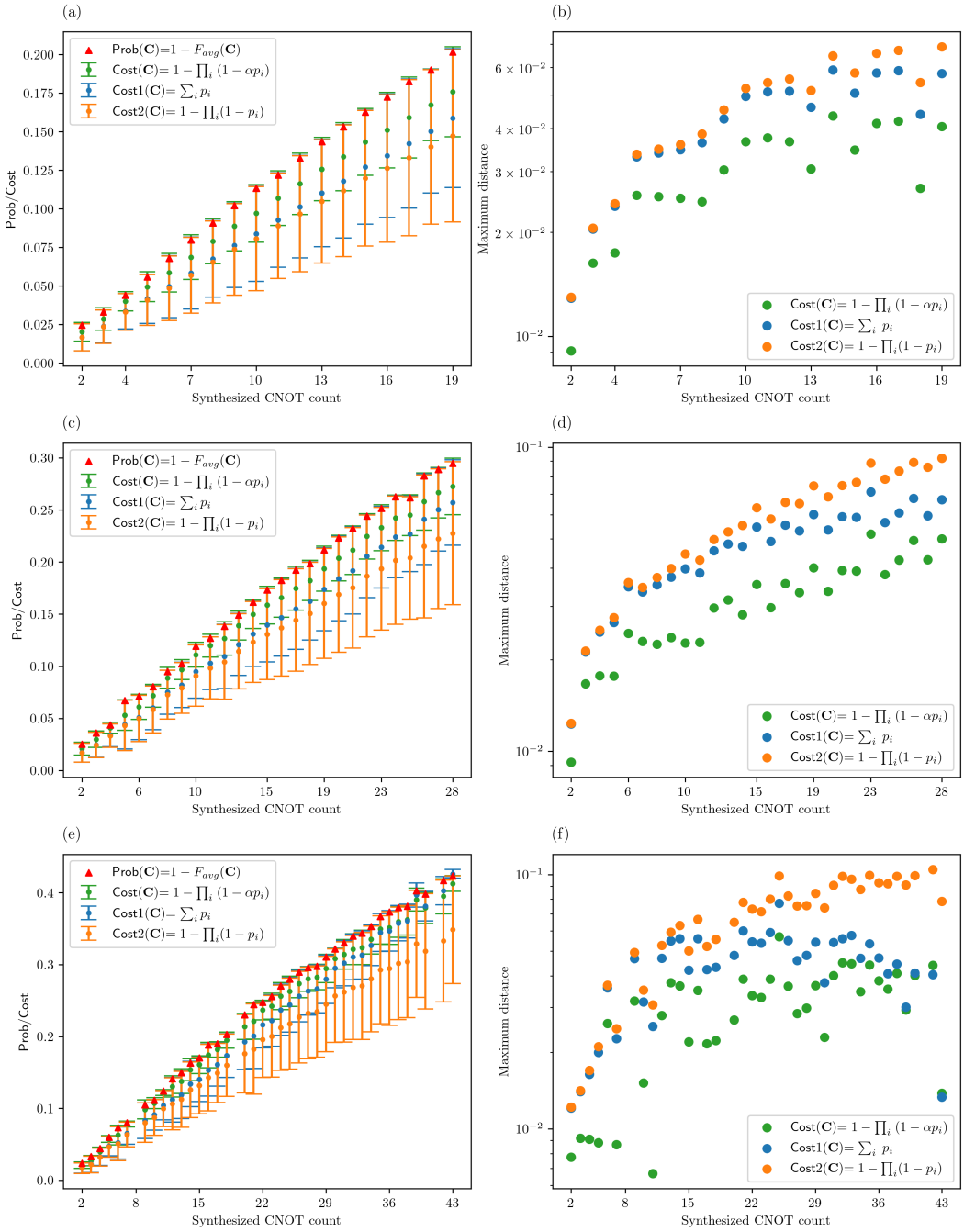


Figure 26: On IBM's fake Guadalupe backend, compare the three cost functions against the error probability of a synthesized CNOT circuit. p_i is the error rate of a noisy CNOT gate, n is the circuit width, $\alpha = 1 + \frac{2^{n-2}-1}{2^n+1}$. In the left column, figures (a), (c), and (e) report results about the synthesized CNOT circuits of widths 5, 6, and 7 respectively. In each figure, for circuits of the same synthesized CNOT count, a y-value corresponds to the average of the error probability or the average of a cost function. The length of an error bar measures the average difference between the error probability and each cost function. The higher the value, the worse the approximation. In the right column, we compare the maximum distance between the error probability and each cost function. Figures (b), (d), and (f) report results about the synthesized CNOT circuits of widths 5, 6, and 7 respectively. Since max|Prob − Cost| has a wide range, we use a logarithmic scale for the y-axis to see all the numbers easily.

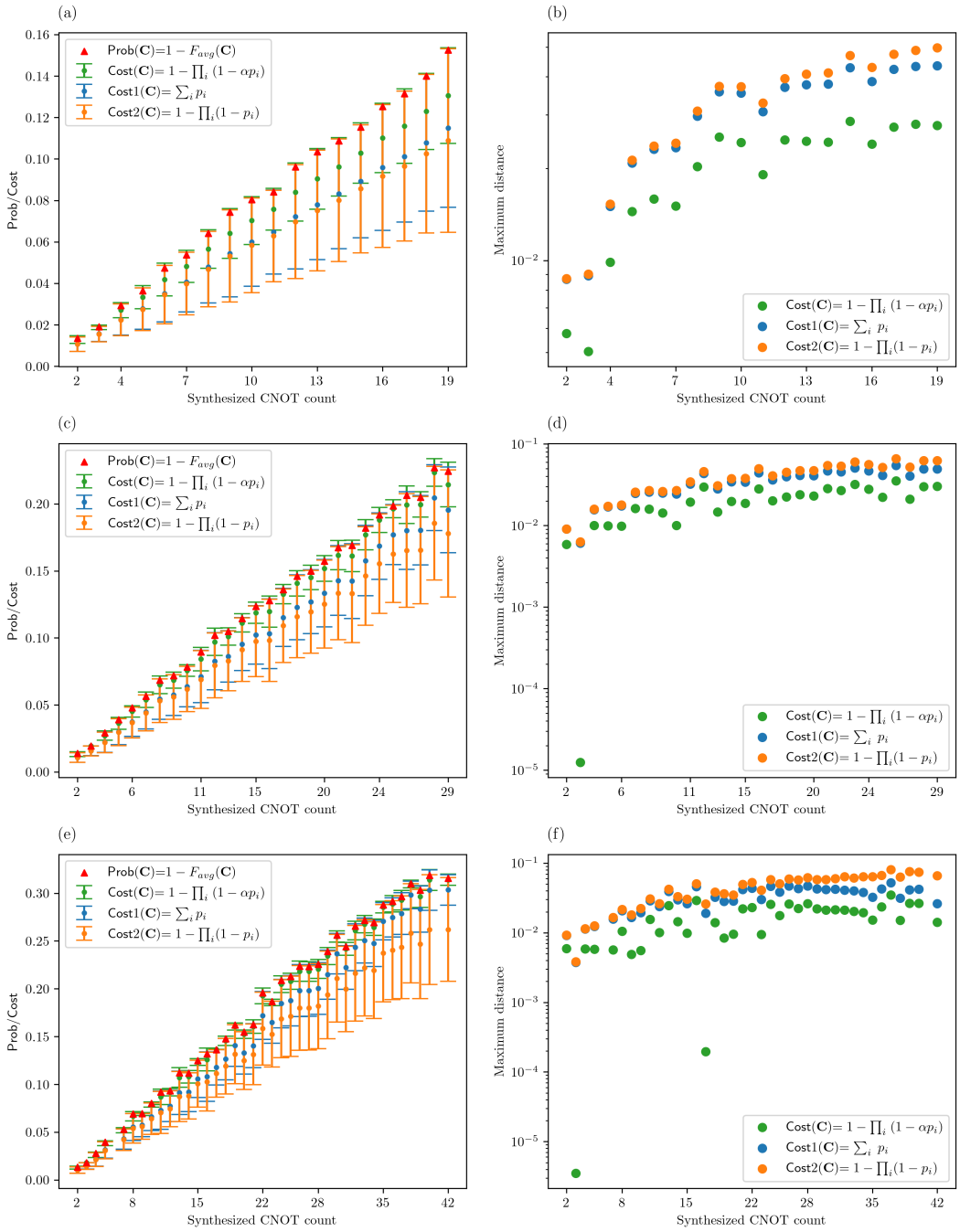


Figure 27: On IBM's fake Cairo backend, compare the three cost functions against the error probability of a synthesized CNOT circuit. p_i is the error rate of a noisy CNOT gate, n is the circuit width, $\alpha = 1 + \frac{2^{n-2}-1}{2^n+1}$. In the left column, figures (a), (c), and (e) report results about the synthesized CNOT circuits of widths 5, 6, and 7 respectively. In each figure, for circuits of the same synthesized CNOT count, a y-value corresponds to the average of the error probability or the average of a cost function. The length of an error bar measures the average difference between the error probability and each cost function. The higher the value, the worse the approximation. In the right column, we compare the maximum distance between the error probability and each cost function. Figures (b), (d), and (f) report results about the synthesized CNOT circuits of widths 5, 6, and 7 respectively. Since max|Prob − Cost| has a wide range, we use a logarithmic scale for the y-axis to see all the numbers easily.

E Compare Different CNOT Synthesis Algorithms on IBM's Fake Backends

E.1 Benchmark on IBM's Fake Nairobi Backend

Circuit Width	Original CNOT Count	Qiskit	ROWCOL	PermRowCol	NAPermRowCol
5	4	4.09 (5.68%)	4.51 (16.54%)	3.99 (3.10%)	3.87
	8	11.22 (23.43%)	11.97 (31.68%)	10.22 (12.43%)	9.09
	16	25.28 (117.93%)	17.02 (46.72%)	13.04 (12.41%)	11.60
	32	55.34 (326.35%)	17.83 (37.37%)	14.11 (8.71%)	12.98
	64	120.14 (896.19%)	19.14 (58.71%)	14.14 (17.25%)	12.06
	128	246.41 (1885.58%)	18.32 (47.62%)	13.76 (10.88%)	12.41
	256	500.59 (4013.31%)	18.40 (51.19%)	13.93 (14.46%)	12.17
	512	1001.44 (8021.98%)	18.89 (53.20%)	13.80 (11.92%)	12.33
	1024	2017.34 (16116.56%)	18.75 (50.72%)	13.62 (9.49%)	12.44
7	4	4.53 (-0.88%)	4.77 (4.38%)	4.70 (2.84%)	4.57
	8	13.05 (4.15%)	16.24 (29.61%)	13.67 (9.10%)	12.53
	16	33.47 (35.01%)	32.01 (29.12%)	26.75 (7.91%)	24.79
	32	79.96 (165.03%)	39.35 (30.43%)	33.64 (11.50%)	30.17
	64	166.18 (433.65%)	41.55 (33.43%)	35.48 (13.94%)	31.14
	128	347.20 (1011.04%)	41.32 (32.22%)	34.68 (10.98%)	31.25
	256	706.32 (2199.97%)	41.72 (35.85%)	34.14 (11.17%)	30.71
	512	1431.80 (4541.17%)	41.75 (35.33%)	34.57 (12.06%)	30.85
	1024	2920.34 (9170.92%)	41.03 (30.25%)	35.12 (11.49%)	31.50

Table 4: IBM's fake Nairobi backend hosts 7 qubits. We benchmark with its 5- and 7-qubit connected subgraph and compare NAPermRowCol with other state-of-the-art CNOT synthesis algorithms in terms of the synthesized CNOT count. Qiskit is short for "Qiskit transpilation at optimization level 3". It implements the SWAP-based heuristic algorithm SABRE. For each row (i.e., the original CNOT count), input 100 randomly generated CNOT circuits to each algorithm listed in the table header, then calculate the average synthesized CNOT count. The value in each bracket shows the percentage difference compared to the results of NAPermRowCol.

Circuit Width	Original CNOT Count	Qiskit		ROWCOL		PermRowCol		NAPermRowCol
5	4	0.0407	(8.21%)	0.0436	(15.93%)	0.0388	(3.29%)	0.0376
	8	0.1075	(25.72%)	0.1123	(31.34%)	0.0968	(13.17%)	0.0855
	16	0.2273	(109.72%)	0.1574	(45.18%)	0.1226	(13.12%)	0.1084
	32	0.4301	(258.69%)	0.1640	(36.78%)	0.1316	(9.72%)	0.1199
	64	0.7079	(527.59%)	0.1743	(54.48%)	0.1328	(17.76%)	0.1128
	128	0.9189	(696.34%)	0.1678	(45.42%)	0.1294	(12.15%)	0.1154
	256	0.9940	(778.12%)	0.1694	(49.64%)	0.1306	(15.41%)	0.1132
	512	1.0000	(780.36%)	0.1721	(51.48%)	0.1277	(12.45%)	0.1136
	1024	1.0000	(770.07%)	0.1729	(50.48%)	0.1267	(10.28%)	0.1149
7	4	0.0437	(-0.23%)	0.0461	(5.45%)	0.0455	(3.95%)	0.0438
	8	0.1222	(4.08%)	0.1516	(29.13%)	0.1289	(9.79%)	0.1174
	16	0.2858	(30.77%)	0.2778	(27.11%)	0.2381	(8.92%)	0.2186
	32	0.5576	(114.27%)	0.3309	(27.16%)	0.2898	(11.37%)	0.2602
	64	0.8167	(205.02%)	0.3446	(28.68%)	0.3035	(13.34%)	0.2678
	128	0.9711	(261.34%)	0.3421	(27.31%)	0.2987	(11.16%)	0.2687
	256	0.9993	(277.49%)	0.3467	(30.98%)	0.2955	(11.62%)	0.2647
	512	1.0000	(277.93%)	0.3467	(31.02%)	0.2975	(12.42%)	0.2646
	1024	1.0000	(270.19%)	0.3418	(26.53%)	0.3006	(11.27%)	0.2701

Table 5: IBM's fake Nairobi backend hosts 7 qubits. We benchmark with its 5- and 7-qubit connected subgraph and compare NAPermRowCol with other state-of-the-art CNOT synthesis algorithms in terms of the Cost metric. For each row (i.e., the original CNOT count), input 100 randomly generated CNOT circuits to each algorithm listed in the table header, then calculate the average Cost. The value in each bracket shows the percentage difference compared to the results of NAPermRowCol.

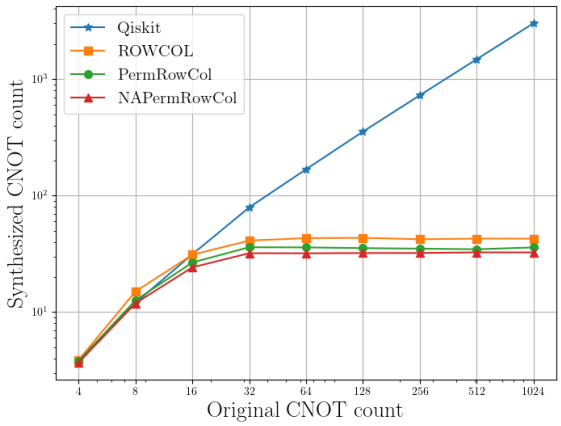
E.2 Benchmark on IBM's Fake Guadalupe Backend

Circuit Width	Original CNOT Count	Qiskit	ROWCOL	PermRowCol	NAPermRowCol
5	4	4.07 (3.04%)	4.34 (9.87%)	3.92 (-0.76%)	3.95
	8	11.08 (23.11%)	12.03 (33.67%)	10.19 (13.22%)	9.00
	16	25.43 (116.43%)	16.94 (44.17%)	13.18 (12.17%)	11.75
	32	56.35 (334.13%)	18.37 (41.53%)	14.23 (9.63%)	12.98
	64	121.11 (893.52%)	18.93 (55.29%)	13.91 (14.11%)	12.19
	128	247.08 (1913.69%)	18.15 (47.92%)	13.69 (11.57%)	12.27
	256	504.43 (3932.21%)	18.50 (47.88%)	14.03 (12.15%)	12.51
	512	1005.74 (8198.18%)	18.75 (54.70%)	13.76 (13.53%)	12.12
	1024	2023.69 (16063.66%)	18.63 (48.80%)	13.89 (10.94%)	12.52
7	4	3.85 (4.90%)	3.86 (5.18%)	3.79 (3.27%)	3.67
	8	11.96 (0.76%)	14.98 (26.20%)	12.56 (5.81%)	11.87
	16	31.29 (29.46%)	31.01 (28.30%)	26.63 (10.18%)	24.17
	32	79.05 (147.26%)	41.06 (28.43%)	35.99 (12.57%)	31.97
	64	168.18 (426.88%)	43.09 (34.99%)	35.87 (12.37%)	31.92
	128	354.01 (1003.52%)	43.33 (35.07%)	35.35 (10.19%)	32.08
	256	726.68 (2161.69%)	42.16 (31.22%)	35.00 (8.93%)	32.13
	512	1483.26 (4462.47%)	42.65 (31.19%)	34.56 (6.31%)	32.51
	1024	3023.32 (9202.52%)	42.69 (31.35%)	35.86 (10.34%)	32.50
16	4	3.98 (1.02%)	5.61 (42.39%)	3.94 (0.00%)	3.94
	8	8.90 (-9.28%)	11.95 (21.81%)	9.24 (-5.81%)	9.81
	16	33.17 (-36.61%)	70.54 (34.80%)	57.85 (10.55%)	52.33
	32	98.06 (-27.84%)	170.99 (25.82%)	146.18 (7.56%)	135.90
	64	252.75 (13.75%)	247.43 (11.35%)	228.18 (2.69%)	222.20
	128	577.35 (134.48%)	274.54 (11.50%)	258.17 (4.85%)	246.23
	256	1261.06 (408.53%)	275.65 (11.16%)	260.65 (5.11%)	247.98
	512	2644.47 (964.69%)	274.35 (10.46%)	262.67 (5.75%)	248.38
	1024	5465.55 (2117.08%)	274.42 (11.32%)	262.28 (6.39%)	246.52

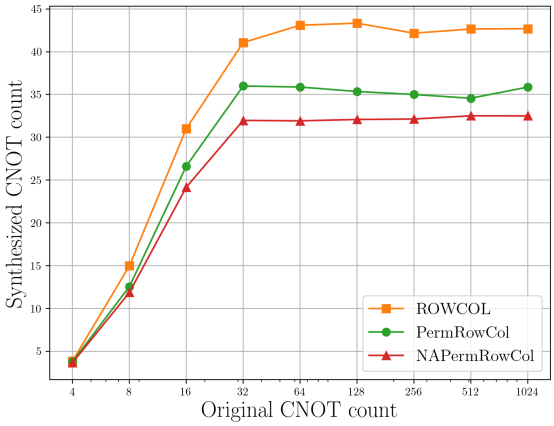
Table 6: IBM's fake Guadalupe backend hosts 16 qubits. We benchmark with its 5-, 7-, and 16-qubit connected subgraph and compare NAPermRowCol with other state-of-the-art CNOT synthesis algorithms in terms of the synthesized CNOT count. Qiskit is short for "Qiskit transpilation at optimization level 3". It implements the SWAP-based heuristic algorithm SABRE. For each row (i.e., the original CNOT count), input 100 randomly generated CNOT circuits to each algorithm listed in the table header, then calculate the average synthesized CNOT count. The value in each bracket shows the percentage difference compared to the results of NAPermRowCol.

Circuit Width	Original CNOT Count	Qiskit		ROWCOL		PermRowCol		NAPermRowCol
5	4	0.0546	(6.56%)	0.0567	(10.59%)	0.0513	(0.06%)	0.0512
	8	0.1379	(30.06%)	0.1425	(34.41%)	0.1231	(16.06%)	0.1060
	16	0.2876	(113.88%)	0.1961	(45.82%)	0.1564	(16.26%)	0.1345
	32	0.5378	(255.14%)	0.2163	(42.82%)	0.1718	(13.46%)	0.1514
	64	0.7987	(467.07%)	0.2175	(54.45%)	0.1640	(16.43%)	0.1408
	128	0.9579	(581.91%)	0.2081	(48.15%)	0.1600	(13.92%)	0.1405
	256	0.9985	(586.23%)	0.2145	(47.43%)	0.1685	(15.83%)	0.1455
	512	1.0000	(622.77%)	0.2148	(55.25%)	0.1601	(15.71%)	0.1384
	1024	1.0000	(592.21%)	0.2152	(48.94%)	0.1645	(13.86%)	0.1445
7	4	0.0520	(5.94%)	0.0514	(4.81%)	0.0506	(3.25%)	0.0491
	8	0.1518	(4.33%)	0.1837	(26.24%)	0.1555	(6.89%)	0.1455
	16	0.3474	(28.99%)	0.3403	(26.35%)	0.3000	(11.40%)	0.2693
	32	0.6571	(92.43%)	0.4266	(24.92%)	0.3846	(12.63%)	0.3415
	64	0.8974	(163.33%)	0.4428	(29.93%)	0.3869	(13.54%)	0.3408
	128	0.9911	(191.90%)	0.4384	(29.12%)	0.3782	(11.38%)	0.3395
	256	0.9999	(198.17%)	0.4291	(27.95%)	0.3734	(11.36%)	0.3354
	512	1.0000	(195.00%)	0.4307	(27.05%)	0.3661	(8.00%)	0.3390
	1024	1.0000	(194.05%)	0.4357	(28.11%)	0.3808	(11.97%)	0.3401
16	4	0.0515	(0.96%)	0.0685	(34.22%)	0.0511	(0.00%)	0.0511
	8	0.1140	(-6.28%)	0.1435	(17.96%)	0.1182	(-2.83%)	0.1216
	16	0.3663	(-25.21%)	0.6028	(23.07%)	0.5273	(7.64%)	0.4898
	32	0.7394	(-11.03%)	0.8983	(8.09%)	0.8601	(3.49%)	0.8310
	64	0.9689	(2.28%)	0.9667	(2.05%)	0.9560	(0.91%)	0.9473
	128	0.9996	(3.82%)	0.9775	(1.52%)	0.9720	(0.95%)	0.9629
	256	1.0000	(3.84%)	0.9780	(1.56%)	0.9726	(0.99%)	0.9630
	512	1.0000	(3.71%)	0.9773	(1.36%)	0.9736	(0.98%)	0.9642
	1024	1.0000	(3.77%)	0.9776	(1.44%)	0.9734	(1.01%)	0.9636

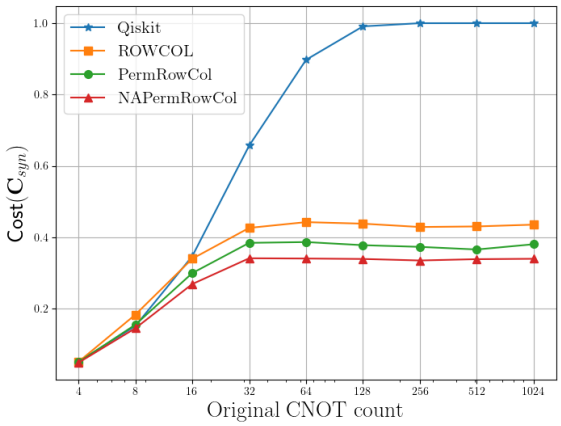
Table 7: IBM's fake Guadalupe backend hosts 16 qubits. We benchmark with its 5-, 7-, and 16-qubit connected subgraph and compare NAPermRowCol with other state-of-the-art CNOT synthesis algorithms in terms of the Cost metric. For each row (i.e., the original CNOT count), input 100 randomly generated CNOT circuits to each algorithm listed in the table header, then calculate the average synthesized CNOT count. The value in each bracket shows the percentage difference compared to the results of NAPermRowCol.



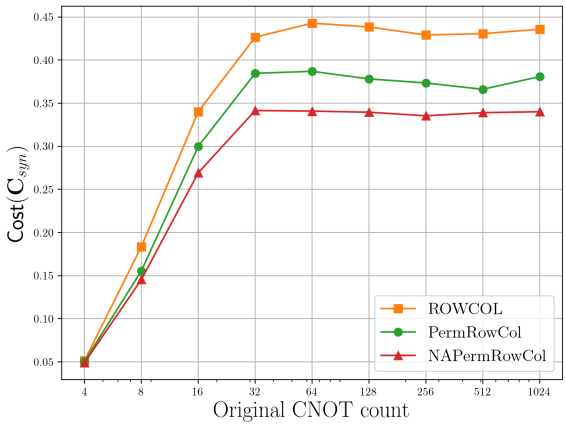
(a) Compare NAPermRowCol with PermRowCol, ROWCOL, and Qiskit in terms of the synthesized CNOT count. Qiskit has the worst scalability when the original CNOT count grows exponentially.



(b) Compare NAPermRowCol with PermRowCol and ROWCOL in terms of the synthesized CNOT count. This is the zoomed-in version of the lefthand side.

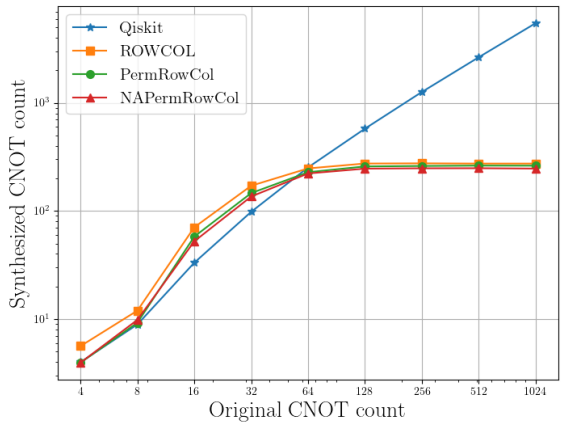


(c) Compare NAPermRowCol with PermRowCol, ROWCOL, and Qiskit in terms of their respective costs. Qiskit has the worst scalability when the original CNOT count grows exponentially.

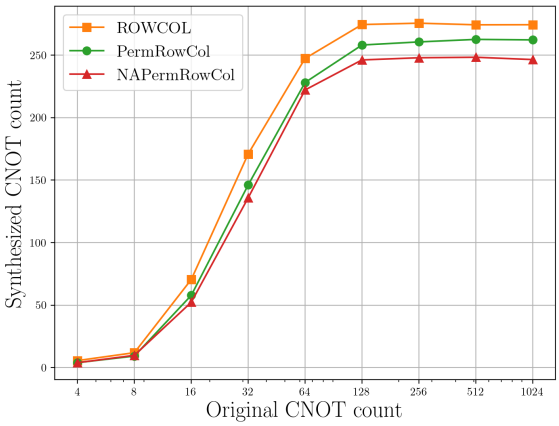


(d) Compare NAPermRowCol with PermRowCol and ROWCOL in terms of their respective costs. This is the zoomed-in version of the lefthand side.

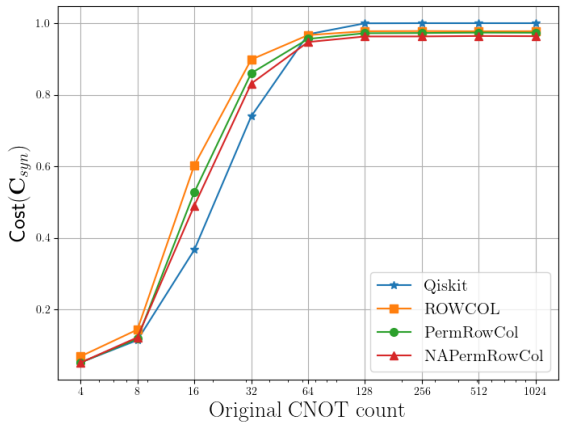
Figure 29: IBM's fake Guadalupe backend hosts 16 qubits. We benchmark with its 7-qubit connected subgraph and compare NAPermRowCol against three state-of-the-art CNOT synthesis algorithms. For each original CNOT count, input 100 randomly generated CNOT circuits to each algorithm, obtain the synthesized CNOT circuits, then average their gate count and the circuit cost. The x-axis in each figure uses a logarithmic scale as the input gate count grows exponentially. The y-axis of Figure 29a uses a logarithmic scale, while the one in Figures 29b to 29d uses a linear scale. Compared to Figures 29a and 29c, Figures 29b and 29d get rid of the data related to Qiskit so that the remaining ones are distributed in a more compact area. They serve as the zoomed-in versions which allow us to compare the performance of NAPermRowCol, PermRowCol, and ROWCOL more closely. For input circuits of more than 16 CNOT counts, NAPermRowCol outperforms other algorithms in terms of the synthesized CNOT count and circuit cost. It demonstrates remarkable scalability when the input circuit size grows exponentially.



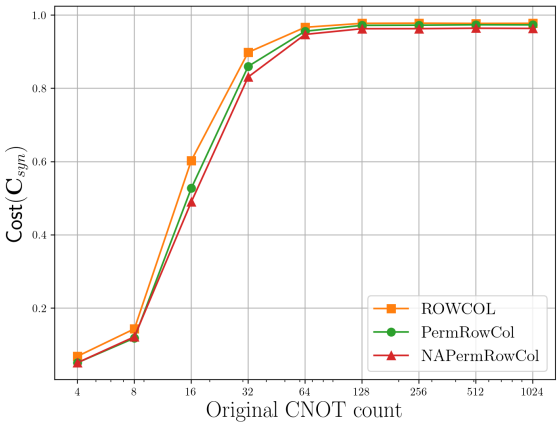
(a) Compare NAPermRowCol with PermRowCol, ROWCOL, and Qiskit in terms of the synthesized CNOT count. Qiskit has the worst scalability when the original CNOT count grows exponentially



(b) Compare NAPermRowCol with PermRowCol and ROWCOL in terms of the synthesized CNOT count. This is the zoomed-in version of the lefthand side.



(c) Compare NAPermRowCol with PermRowCol, ROWCOL, and Qiskit in terms of their respective costs. Qiskit has the worst scalability when the original CNOT count grows exponentially.



(d) Compare NAPermRowCol with PermRowCol and ROWCOL in terms of their respective costs. This is the zoomed-in version of the lefthand side.

Figure 30: IBM's fake Guadalupe backend hosts 16 qubits. We benchmark with its 16-qubit connected subgraph and compare NAPermRowCol against three state-of-the-art CNOT synthesis algorithms. For each original CNOT count, input 100 randomly generated CNOT circuits to each algorithm, obtain the synthesized CNOT circuits, then average their gate count and the circuit cost. The x-axis in each figure uses a logarithmic scale as the input gate count grows exponentially. The y-axis of Figure 30a uses a logarithmic scale, while the one in Figures 30b to 30d uses a linear scale. Compared to Figures 30a and 30c, Figures 30b and 30d get rid of the data related to Qiskit so that the remaining ones are distributed in a more compact area. They serve as the zoomed-in versions which allow us to compare the performance of NAPermRowCol, PermRowCol, and ROWCOL more closely. For input circuits of more than 64 CNOT counts, NAPermRowCol outperforms other algorithms in terms of the synthesized CNOT count and circuit cost. It demonstrates remarkable scalability when the input circuit size grows exponentially.

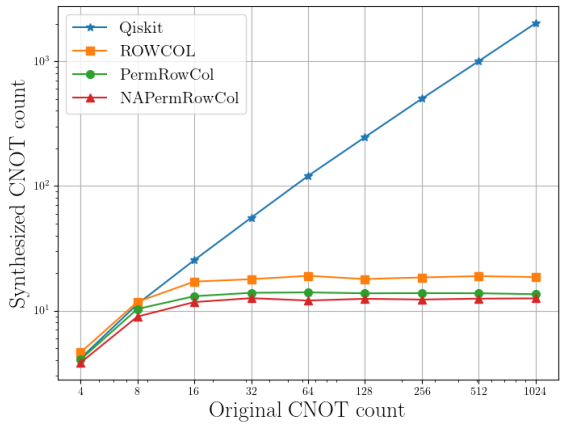
E.3 Benchmark on IBM's Fake Cairo Backend

Circuit Width	Original CNOT Count	Qiskit	ROWCOL	PermRowCol	NAPermRowCol
5	4	4.13 (8.12%)	4.61 (20.68%)	4.05 (6.02%)	3.82
	8	11.28 (26.03%)	11.75 (31.28%)	10.30 (15.08%)	8.95
	16	25.41 (117.18%)	17.09 (46.07%)	13.04 (11.45%)	11.70
	32	55.58 (341.81%)	17.85 (41.89%)	13.89 (10.41%)	12.58
	64	120.31 (899.25%)	19.00 (57.81%)	14.00 (16.28%)	12.04
	128	246.29 (1881.42%)	17.91 (44.09%)	13.76 (10.70%)	12.43
	256	501.19 (3988.01%)	18.45 (50.49%)	13.80 (12.56%)	12.26
	512	1002.96 (7942.98%)	18.89 (51.48%)	13.77 (10.43%)	12.47
	1024	2021.27 (16031.44%)	18.56 (48.12%)	13.54 (8.06%)	12.53
7	4	3.85 (3.49%)	3.85 (3.49%)	3.82 (2.69%)	3.72
	8	11.86 (3.58%)	14.60 (27.51%)	12.53 (9.43%)	11.45
	16	30.90 (26.33%)	31.52 (28.86%)	26.34 (7.69%)	24.46
	32	76.55 (152.39%)	40.04 (32.01%)	34.34 (13.22%)	30.33
	64	164.84 (420.66%)	42.13 (33.07%)	35.48 (12.07%)	31.66
	128	342.74 (973.07%)	41.74 (30.68%)	35.19 (10.18%)	31.94
	256	708.43 (2156.87%)	41.70 (32.84%)	33.54 (6.85%)	31.39
	512	1452.64 (4495.51%)	40.42 (27.87%)	34.83 (10.19%)	31.61
	1024	2925.63 (8982.99%)	40.28 (25.05%)	34.12 (5.93%)	32.21
16	4	3.98 (1.53%)	4.40 (12.24%)	3.94 (0.51%)	3.92
	8	8.82 (-2.97%)	9.65 (6.16%)	8.96 (-1.43%)	9.09
	16	32.69 (-38.10%)	60.23 (14.05%)	55.92 (5.89%)	52.81
	32	102.45 (-20.72%)	153.30 (18.63%)	135.98 (5.23%)	129.22
	64	264.28 (21.73%)	245.55 (13.10%)	226.74 (4.44%)	217.11
	128	603.25 (141.35%)	278.36 (11.37%)	260.97 (4.41%)	249.95
	256	1294.44 (416.74%)	275.55 (10.00%)	264.12 (5.44%)	250.50
	512	2694.88 (972.50%)	276.54 (10.06%)	263.68 (4.94%)	251.27
	1024	5527.37 (2111.39%)	274.74 (9.92%)	261.06 (4.44%)	249.95

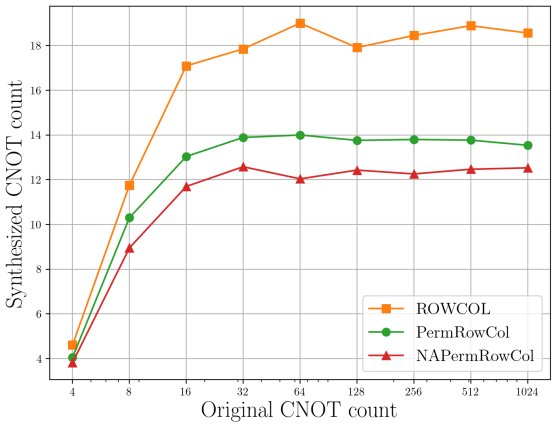
Table 8: IBM's fake Cairo backend hosts 27 qubits. We benchmark with its 5-, 7-, and 16-qubit connected subgraph and compare NAPermRowCol with other state-of-the-art CNOT synthesis algorithms in terms of the synthesized CNOT count. Qiskit is short for "Qiskit transpilation at optimization level 3". It implements the SWAP-based heuristic algorithm SABRE. For each row (i.e., the original CNOT count), input 100 randomly generated CNOT circuits to each algorithm listed in the table header, then calculate the average synthesized CNOT count. The value in each bracket shows the percentage difference compared to the results of NAPermRowCol.

Circuit Width	Original CNOT Count	Qiskit		ROWCOL		PermRowCol		NAPermRowCol
5	4	0.0547	(8.11%)	0.0609	(20.35%)	0.0559	(10.47%)	0.0506
	8	0.1605	(34.55%)	0.1561	(30.82%)	0.1451	(21.60%)	0.1193
	16	0.3100	(105.96%)	0.2211	(46.86%)	0.1713	(13.81%)	0.1505
	32	0.5277	(254.01%)	0.2188	(46.77%)	0.1709	(14.64%)	0.1491
	64	0.7903	(435.71%)	0.2315	(56.92%)	0.1774	(20.24%)	0.1475
	128	0.9433	(547.99%)	0.2085	(43.24%)	0.1666	(14.44%)	0.1456
	256	0.9956	(594.40%)	0.2147	(49.77%)	0.1666	(16.20%)	0.1434
	512	1.0000	(547.47%)	0.2398	(55.27%)	0.1807	(16.97%)	0.1544
	1024	1.0000	(523.26%)	0.2386	(48.74%)	0.1808	(12.71%)	0.1604
7	4	0.0548	(3.36%)	0.0549	(3.59%)	0.0542	(2.15%)	0.0530
	8	0.1632	(9.82%)	0.1933	(30.11%)	0.1660	(11.72%)	0.1486
	16	0.3549	(28.35%)	0.3618	(30.86%)	0.3077	(11.30%)	0.2765
	32	0.6530	(96.39%)	0.4239	(27.49%)	0.3849	(15.76%)	0.3325
	64	0.8849	(160.36%)	0.4419	(30.01%)	0.3914	(15.15%)	0.3399
	128	0.9854	(189.61%)	0.4356	(28.01%)	0.3792	(11.45%)	0.3403
	256	0.9997	(185.68%)	0.4611	(31.77%)	0.3972	(13.51%)	0.3500
	512	1.0000	(194.61%)	0.4340	(27.86%)	0.3854	(13.53%)	0.3394
	1024	1.0000	(192.17%)	0.4313	(26.01%)	0.3769	(10.11%)	0.3423
16	4	0.0575	(0.96%)	0.0632	(11.03%)	0.0573	(0.56%)	0.0569
	8	0.1257	(-0.32%)	0.1339	(6.14%)	0.1264	(0.17%)	0.1262
	16	0.4049	(-17.47%)	0.5615	(14.46%)	0.5343	(8.91%)	0.4906
	32	0.7800	(-3.71%)	0.8810	(8.75%)	0.8478	(4.65%)	0.8101
	64	0.9811	(3.71%)	0.9747	(3.03%)	0.9647	(1.97%)	0.9460
	128	0.9998	(3.99%)	0.9809	(2.02%)	0.9757	(1.48%)	0.9614
	256	1.0000	(3.61%)	0.9810	(1.64%)	0.9777	(1.30%)	0.9651
	512	1.0000	(3.37%)	0.9837	(1.68%)	0.9790	(1.20%)	0.9674
	1024	1.0000	(4.11%)	0.9775	(1.76%)	0.9723	(1.22%)	0.9605

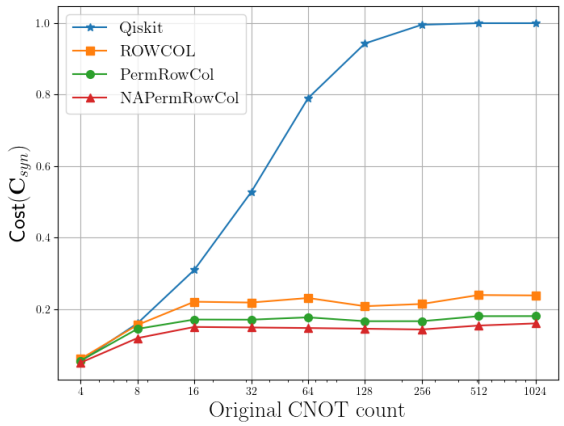
Table 9: IBM's fake Cairo backend hosts 27 qubits. We benchmark with its 5-, 7-, and 16-qubit connected subgraph and compare NAPermRowCol with other state-of-the-art CNOT synthesis algorithms in terms of the Cost metric. For each row (i.e., the original CNOT count), input 100 randomly generated CNOT circuits to each algorithm listed in the table header, then calculate the average synthesized CNOT count. The value in each bracket shows the percentage difference compared to the results of NAPermRowCol.



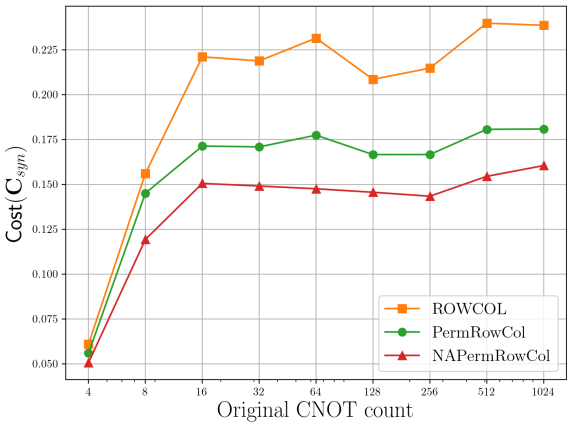
(a) Compare NAPermRowCol with PermRowCol, ROWCOL, and Qiskit in terms of the synthesized CNOT count. Qiskit has the worst scalability when the original CNOT count grows exponentially.



(b) Compare NAPermRowCol with PermRowCol and ROWCOL in terms of the synthesized CNOT count. This is the zoomed-in version of the lefthand side.

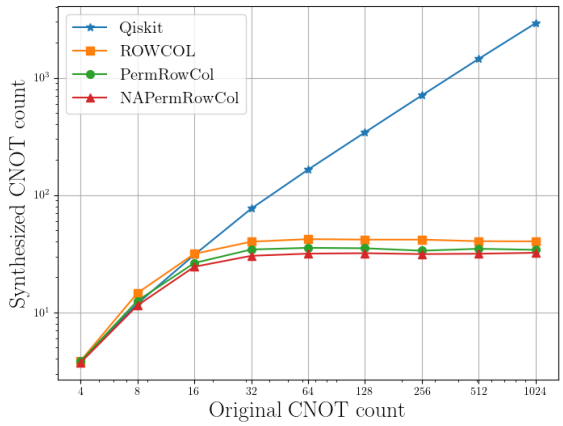


(c) Compare NAPermRowCol with PermRowCol, ROWCOL, and Qiskit in terms of their respective costs. Qiskit has the worst scalability when the original CNOT count grows exponentially.

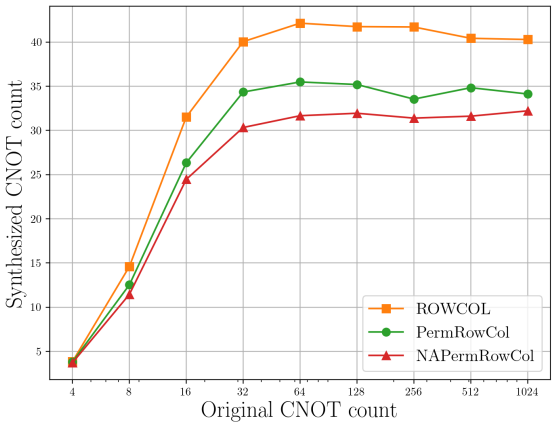


(d) Compare NAPermRowCol with PermRowCol and ROWCOL in terms of their respective costs. This is the zoomed-in version of the lefthand side.

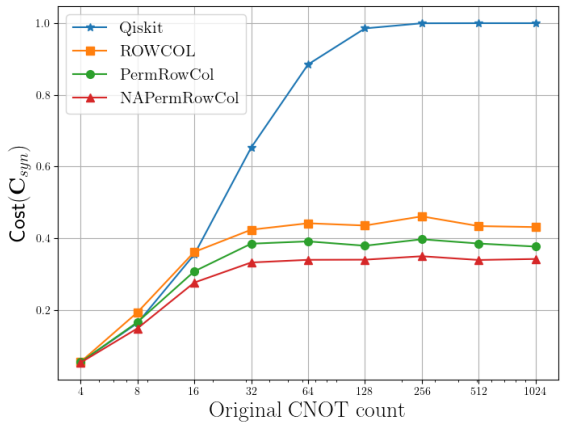
Figure 31: IBM's fake Cairo backend hosts 27 qubits. We benchmark with its 5-qubit connected subgraph and compare NAPermRowCol against three state-of-the-art CNOT synthesis algorithms. For each original CNOT count, input 100 randomly generated CNOT circuits to each algorithm, obtain the synthesized CNOT circuits, then average their gate count and the circuit cost. The x-axis in each figure uses a logarithmic scale as the input gate count grows exponentially. The y-axis of Figure 31a uses a logarithmic scale, while the one in Figures 31b to 31d uses a linear scale. Compared to Figures 31a and 31c, Figures 31b and 31d get rid of the data related to Qiskit so that the remaining ones are distributed in a more compact area. They serve as the zoomed-in versions which allow us to compare the performance of NAPermRowCol, PermRowCol, and ROWCOL more closely. For all input circuits of different CNOT counts, NAPermRowCol outperforms other algorithms in terms of the synthesized CNOT count and circuit cost. It demonstrates remarkable scalability when the input circuit size grows exponentially.



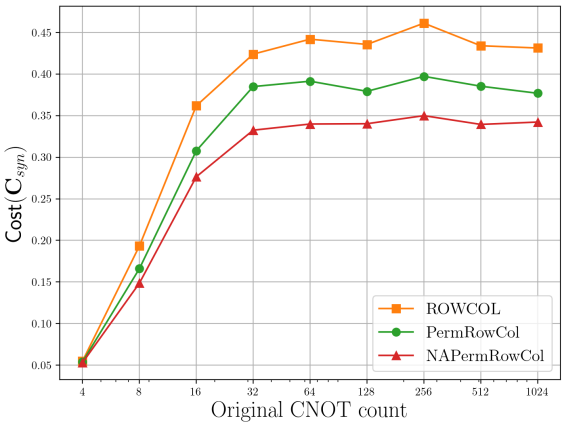
(a) Compare NAPermRowCol with PermRowCol, ROWCOL, and Qiskit in terms of the synthesized CNOT count. Qiskit has the worst scalability when the original CNOT count grows exponentially.



(b) Compare NAPermRowCol with PermRowCol and ROWCOL in terms of the synthesized CNOT count. This is the zoomed-in version of the lefthand side.

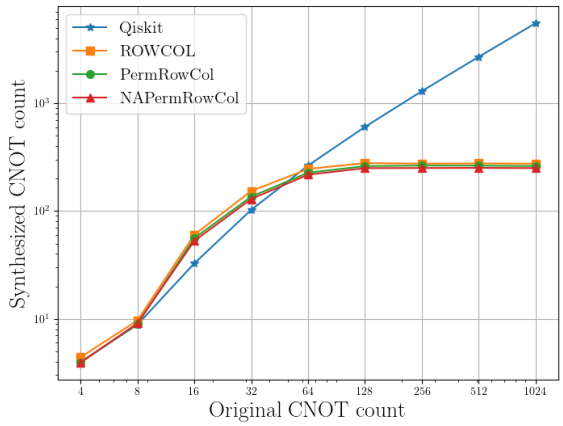


(c) Compare NAPermRowCol with PermRowCol, ROWCOL, and Qiskit in terms of their respective costs. Qiskit has the worst scalability when the original CNOT count grows exponentially.

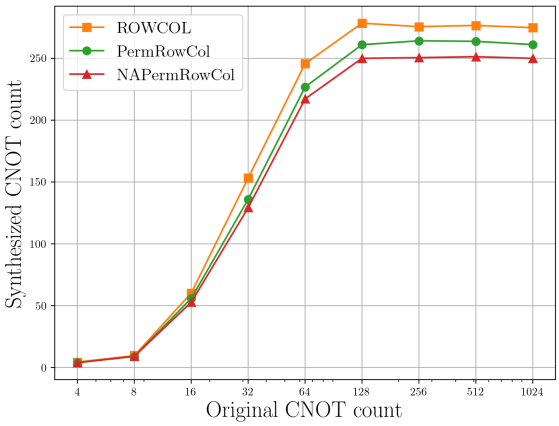


(d) Compare NAPermRowCol with PermRowCol and ROWCOL in terms of their respective costs. This is the zoomed-in version of the lefthand side.

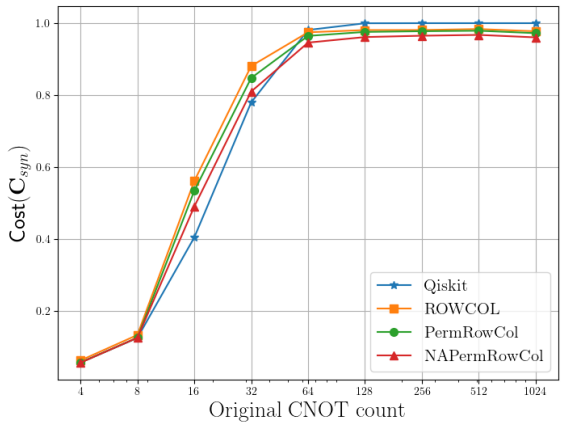
Figure 32: IBM's fake Cairo backend hosts 27 qubits. We benchmark with its 7-qubit connected subgraph and compare NAPermRowCol against three state-of-the-art CNOT synthesis algorithms. For each original CNOT count, input 100 randomly generated CNOT circuits to each algorithm, obtain the synthesized CNOT circuits, then average their gate count and the circuit cost. The x-axis in each figure uses a logarithmic scale as the input gate count grows exponentially. The y-axis of Figure 32a uses a logarithmic scale, while the one in Figures 32b to 32d uses a linear scale. Compared to Figures 32a and 32c, Figures 32b and 32d get rid of the data related to Qiskit so that the remaining ones are distributed in a more compact area. They serve as the zoomed-in versions which allow us to compare the performance of NAPermRowCol, PermRowCol, and ROWCOL more closely. For all input circuits of different CNOT counts, NAPermRowCol outperforms other algorithms in terms of the synthesized CNOT count and circuit cost. It demonstrates remarkable scalability when the input circuit size grows exponentially.



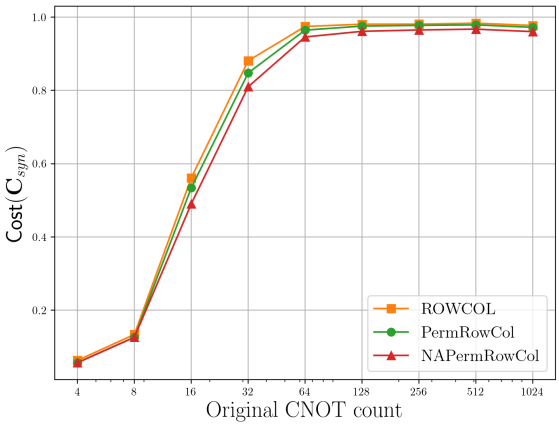
(a) Compare NAPermRowCol with PermRowCol, ROWCOL, and Qiskit in terms of the synthesized CNOT count. Qiskit has the worst scalability when the original CNOT count grows exponentially.



(b) Compare NAPermRowCol with PermRowCol and ROWCOL in terms of the synthesized CNOT count. This is the zoomed-in version of the lefthand side.

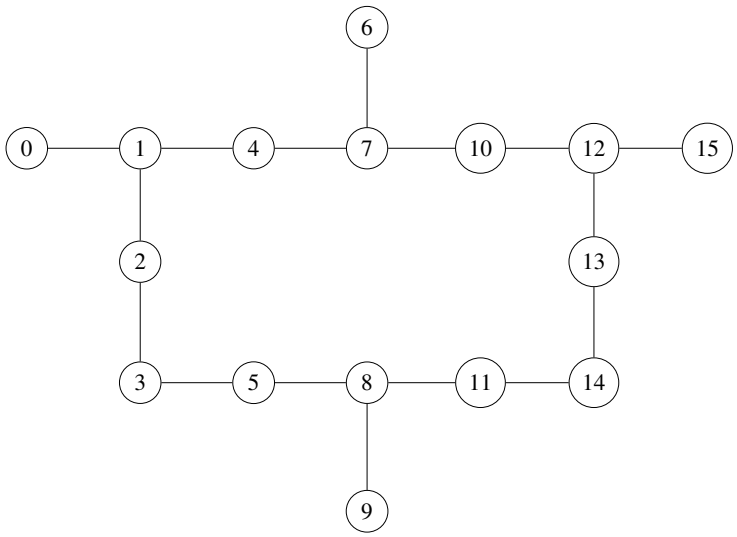


(c) Compare NAPermRowCol with PermRowCol, ROWCOL, and Qiskit in terms of their respective costs. Qiskit has the worst scalability when the original CNOT count grows exponentially.



(d) Compare NAPermRowCol with PermRowCol and ROWCOL in terms of their respective costs. This is the zoomed-in version of the lefthand side.

Figure 33: IBM's fake Cairo backend hosts 27 qubits. We benchmark with its 16-qubit connected subgraph and compare NAPermRowCol against three state-of-the-art CNOT synthesis algorithms. For each original CNOT count, input 100 randomly generated CNOT circuits to each algorithm, obtain the synthesized CNOT circuits, then average their gate count and the circuit cost. The x-axis in each figure uses a logarithmic scale as the input gate count grows exponentially. The y-axis of Figure 33a uses a logarithmic scale, while the one in Figures 33b to 33d uses a linear scale. Compared to Figures 33a and 33c, Figures 33b and 33d get rid of the data related to Qiskit so that the remaining ones are distributed in a more compact area. They serve as the zoomed-in versions which allow us to compare the performance of NAPermRowCol, PermRowCol, and ROWCOL more closely. For input circuits of more than 64 CNOT counts, NAPermRowCol outperforms other algorithms in terms of the synthesized CNOT count and circuit cost. It demonstrates remarkable scalability when the input circuit size grows exponentially.

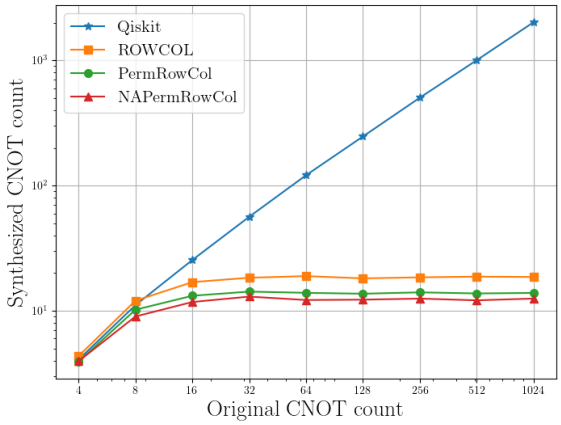


(a) Each vertex corresponds to a physical qubit. Each edge represents a CNOT gate that can be performed on the qubits corresponding to its endpoints.

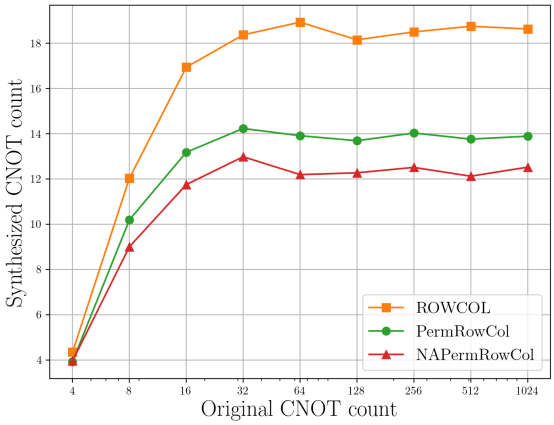
Edge	Edge Weight
(0, 1)	0.009690
(1, 2)	0.015158
(1, 4)	0.007311
(2, 3)	0.013654
(3, 5)	0.012821
(4, 7)	0.011911
(5, 8)	0.008868
(6, 7)	0.006946
(7, 10)	0.006762
(8, 9)	0.012718
(8, 11)	0.009196
(10, 12)	0.019895
(11, 14)	0.010583
(12, 13)	0.007202
(12, 15)	0.007804
(13, 14)	0.012091

(b) For $e = (u, v) \in E_G$, $\omega_G(e)$ is the error rate of coupling physical qubits u and v.

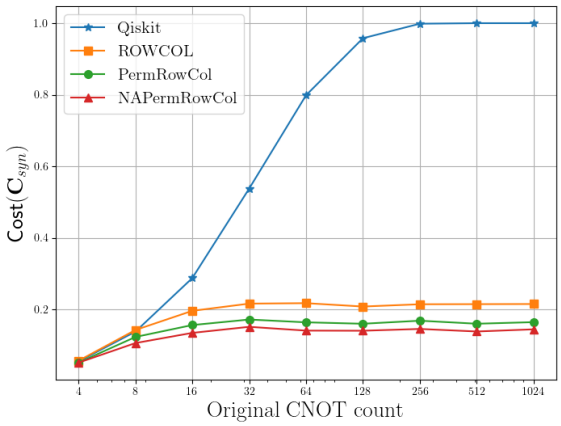
Figure 24: $G = (V_G, E_G, \omega_G)$ is the connectivity graph of IBM's fake Guadalupe backend. It is an undirected edge-weighted connected graph. $|V_G| = 16$, $\omega_G : E_G \to \{x \in \mathbb{R};\ 0 \leq x < 1\}$.



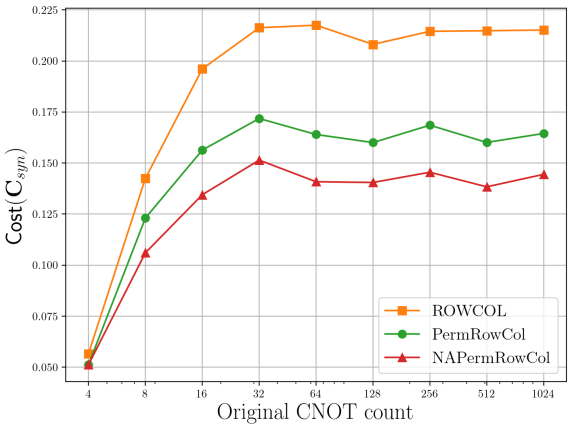
(a) Compare NAPermRowCol with PermRowCol, ROWCOL, and Qiskit in terms of the synthesized CNOT count. Qiskit has the worst scalability when the original CNOT count grows exponentially.



(b) Compare NAPermRowCol with PermRowCol and ROWCOL in terms of the synthesized CNOT count. This is the zoomed-in version of the lefthand side.



(c) Compare NAPermRowCol with PermRowCol, ROWCOL, and Qiskit in terms of their respective costs. Qiskit has the worst scalability when the original CNOT count grows exponentially.



(d) Compare NAPermRowCol with PermRowCol and ROWCOL in terms of their respective costs. This is the zoomed-in version of the lefthand side.

Figure 28: IBM's fake Guadalupe backend hosts 16 qubits. We benchmark with its 5-qubit connected subgraph and compare NAPermRowCol against three state-of-the-art CNOT synthesis algorithms. For each original CNOT count, input 100 randomly generated CNOT circuits to each algorithm, obtain the synthesized CNOT circuits, then average their gate count and the circuit cost. The x-axis in each figure uses a logarithmic scale as the input gate count grows exponentially. The y-axis of Figure 28a uses a logarithmic scale, while the one in Figures 28b to 28d uses a linear scale. Compared to Figures 28a and 28c, Figures 28b and 28d get rid of the data related to Qiskit so that the remaining ones are distributed in a more compact area. They serve as the zoomed-in versions which allow us to compare the performance of NAPermRowCol, PermRowCol, and ROWCOL more closely. For all input circuits of different CNOT counts, NAPermRowCol outperforms other algorithms in terms of the synthesized CNOT count and circuit cost. It demonstrates remarkable scalability when the input circuit size grows exponentially.