UNIVERSITY OF CALIFORNIA

Los Angeles

Layout Synthesis for Quantum Computing

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Computer Science

by

Bochen Tan

2024

ABSTRACT OF THE DISSERTATION

Layout Synthesis for Quantum Computing

by

Bochen Tan
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 2024
Professor Jingsheng Jason Cong, Chair

Quantum computers are expected to surpass conventional computers in tasks like integer factoring and quantum system simulation. In quantum programming, entangling operations between qubits are crucial. However, the program qubits are mapped to physical qubits within a quantum computing architecture that often features restricted connectivity, meaning that entangling operations can only be applied between specific pairs of qubits. As a result, it is essential to determine the map from program qubits to physical qubits throughout the computation. Additionally, it may be necessary to introduce new operations to ensure the quantum program conforms to the connectivity constraints. This challenge is known as layout synthesis for quantum computing. In this dissertation, we explore layout synthesis for quantum architectures, focusing on static architectures primarily based on superconducting circuits, as well as dynamic architectures based on neutral atoms. Given that fault tolerance is required for large-scale quantum computing, we also investigate program synthesis for a promising fault-tolerant quantum architecture based on surface codes.

The dissertation of Bochen Tan is approved.

Miryung Kim

Jens Palsberg

Chee Wei Wong

Jingsheng Jason Cong, Committee Chair

University of California, Los Angeles

2024

To my family

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I am extremely grateful to my PhD advisor, Prof. Jason Cong, for guiding the direction of my research and providing countless practical suggestions.

I would alos like to thank the other members in my PhD committee, Prof. Miryung Kim, Prof. Jens Palsberg, and Prof. Chee Wei Wong, for shaping this dissertation.

I am also grateful to other colleagues listed below.

• Dolev Bluvstein, Prof. Mikhail D. Lukin, Dr. Harry Hengyun Zhou, and Wan-Hsuan Lin for collaboration on various topics in neutral atom based quantum computing, which yields Chapter 6 and Chapter 7;

• Wan-Hsuan Lin, Jason Kimko, Dr. Nikolaj S. Bjørner, Michael Lo, and Chengdi Cao for collaboration on the layout synthesis for static quantum architectures [LKT23];

• Prof. Murphy Yuezhen Niu, Craig Gidney, Dr. Noah Shutty, and Dr. Sergio Boixo for guidance during a research project at Google on fault-tolerant quantum computing [TNG24], which yields Chapter 10;

• Dr. Hanrui Wang, Pengyu Liu, Yilian Liu, Dr. Jiaqi Gu, Prof. David Z. Pan, Prof. Umut A. Acar, and Prof. Song Han for collaboration on layout synthesis for neutral atom arrays [WTL24, WLT24];

• Wan-Hsuan Lin and Prof. Murphy Yuezhen Niu for collaboration on domain-specific quantum architectures [LTN22];

• Hanyu Wang for collaboration on logic synthesis of quantum circuits [WTC24a, WTC24b];

• Shuohao Ping for collaboration on exact binary matrix factorization [TPC24], which yields Chapter 8;

• Dr. Atefeh Sohrabizadeh, Wan-Hsuan Lin, Madelyn Cain, Dr. Sheng-Tao Wang, and Prof. Mikhail D. Lukin for collaboration on the graph neural network based performance prediction for quantum optimization of maximum independent set [SLT24];

• Prof. Guojie Luo and Prof. Hong Guo at Peking University for the recommendation to the PhD program;

• Dr. Yunong Shi and Dr. Eric M. Kessler for guidance during an internship at Amazon Web Services;

• Dr. Kevin Krsulich, Dr. Thomas Alexander, and Dr. Michael B. Healy for guidance during an internship at International Business Machines;

• Prof. Weiwen Jiang for the invitation to an ICCAD '21 special session [TC21a] and the invitation to a DAC '22 tutorial;

• Dr. Rasit O. Topaloglu for the invitation of contributing a chapter to the book *Design Automation of Quantum Computers* [TC23] and the editorial efforts;

• Prof. Giovanni De Micheli for the invitation to a DATE '24 focus session [TPC24];

• Dr. Zhiding Liang for the invitation to an ICCAD '24 special session [SLT24];

• Staffs at the UCLA computer science department, especially Alexandra Luong, Joseph Brown, and Helen Tran, for the assistance during my course of study;

• Marci Baun for the editorial suggestions on various manuscripts;

• Co-workers in the VAST lab during my time, i.e. Dr. Young-Kyu Choi, Dr. Jie Wang, Dr. Weikang Qiao, Dr. Zhe Chen, Dr. Yuze Chi, Dr. Atefeh Sohrabizadeh, Karl Marrett, Dr. Licheng Guo, Jason Lau, Michael Lo, Mrunal Patel, Suhail Basalama, Dr. Linghao Song, Wan-Hsuan Lin, Stéphane Pouget, Neha Prakriya, Jason Kimko, Chengdi Cao, Dr. Lorenzo Ferretti, Zijian Ding, Zifan He, Jake Ke, Cristian Tirelli, and Hanyu Wang.

• Others that provided valuable advice including Prof. Peipei Zhou, Prof. Chen Zhang, and Prof. Zhiru Zhang.

I acknowledge funding from Intel, NEC, Synopsys, and AWS under the Center for Domain-Specific Computing (CDSC) Industrial Partnership Program, and additionally, NSF grant 2313083.

VITA

2015–2019 B.S. Electrical Engineering, Peking University.

2019–2022 M.S. Computer Science, University of California, Los Angeles.

PUBLICATIONS

B. Tan and J. Cong. "Optimality Study of Existing Quantum Computing Layout Synthesis Tools." *IEEE Transactions on Computers* **70**(9):1363–1373, 2021.

B. Tan and J. Cong. "Optimal Layout Synthesis for Quantum Computing." ICCAD 2020.

B. Tan and J. Cong. "Optimal Qubit Mapping with Simultaneous Gate Absorption." ICCAD 2021.

W.-H. Lin, **B. Tan**, M. Y. Niu, J. Kimko, and J. Cong. "Domain-Specific Quantum Architecture Optimization." *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, **12**(3):624–637, 2022.

B. Tan and J. Cong. "Layout Synthesis for Near-Term Quantum Computing: Gap Analysis and Optimal Solution." In R. O. Topaloglu, editor, *Design Automation of Quantum Computers*, pp. 25–40, Springer International Publishing, Cham, 2023.

B. Tan, D. Bluvstein, M. D. Lukin, and J. Cong. "Qubit Mapping for Reconfigurable Atom Arrays." ICCAD 2022.

W.-H. Lin, J. Kimko, **B. Tan**, N. Bjørner, and J. Cong. "Scalable Optimal Layout Synthesis for NISQ Quantum Processors." DAC 2023.

D. B. Tan, D. Bluvstein, M. D. Lukin, and J. Cong. "Compiling Quantum Circuits for Dynamically Field-Programmable Neutral Atoms Array Processors." *Quantum* **8**:1281, 2024.

H. Wang*, **D. B. Tan***, P. Liu, Y. Liu, J. Gu, J. Cong, and S. Han "Q-Pilot: Field Programmable Qubit Array Compilation with Flying Ancillas." DAC 2024.

H. Wang, P. Liu, **D. B. Tan**, Y. Liu, J. Gu, D. Z. Pan, J. Cong, U. A. Acar, and S. Han "Atomique: A Quantum Compiler for Reconfigurable Neutral Atom Arrays." ISCA 2024.

D. B. Tan, M. Y. Niu, and C. Gidney. "A SAT Scalpel for Lattice Surgery: Representation and Synthesis of Subroutines for Surface-Code Fault-Tolerant Quantum Computing." ISCA 2024.

D. B. Tan, S. Ping, and J. Cong. "Depth-Optimal Addressing of 2D Qubit Array with 1D Controls Based on Exact Binary Matrix Factorization." DATE 2024.

D. B. Tan, W.-H. Lin, and J. Cong. "Compilation for Dynamically Field-Programmable Qubit Arrays with Efficient and Provably Near-Optimal Scheduling." ASP-DAC 2025.

H. Wang, **D. B. Tan**, and J. Cong. "Quantum State Preparation Circuit Optimization Exploiting Don't Cares." ICCAD 2024.

CHAPTER 1

Introduction

Over the past few decades, information technology has greatly benefited from Moore's law and Dennard scaling: as semiconductor transistors shrink, more can be packed into a computing chip, enabling larger chips that consume the same amount of power per unit area while also running faster. However, as transistor sizes approach the physical limits, drastic innovations are required to continue advancing computing capabilities.

One major direction is quantum computing, as there are problems—such as integer factoring—which are widely believed to be unsolvable with polynomially many classical operations but can be solved with polynomially many quantum operations [Sho94]. This has driven rapid advancements in scaling up quantum computing hardware which leads to the focus of this dissertation: layout synthesis for quantum computing.

At a high level, quantum registers in real quantum computers have limited connectivity—a constraint too nuanced for programmers to handle directly. Layout synthesis addresses this challenge by determining how quantum data move within the quantum computer, ensuring that quantum algorithms can be executed correctly.

1.1 Concepts in Gate-Based Quantum Computing

For the purposes of this dissertation, only basic concepts in quantum computing are required. Readers seeking more in-depth background can refer to textbooks on quantum computing and quantum information, such as [NC10].

1.1.1 Quantum States

A *qubit* (quantum bit) is in a quantum state $|\psi\rangle$ represented by a vector with unit norm in the two-dimensional complex vector space

$$|\psi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \text{ with } |\alpha|^2 + |\beta|^2 = 1. \tag{1.1}$$

We can decompose the vector with the standard basis

$$|\psi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \alpha \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \beta \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \alpha|0\rangle + \beta|1\rangle. \tag{1.2}$$

We use the *ket* notation $|0\rangle$ and $|1\rangle$ to represent *basis states*, which correspond to the basis vectors. The coefficients of basis states, α and β, are dubbed *amplitudes*. In contrast, a bit is either 0 or 1, but cannot be a linear combination, i.e., *superposition*, of 0 and 1.

However, we cannot directly access the amplitudes of a quantum state in general. The only way we can extract information from a qubit is by *measurements*. When we prepare $\alpha|0\rangle + \beta|1\rangle$ and measure it in basis $\{|0\rangle, |1\rangle\}$, we always receive either 0 or 1. The probability of receiving 0 and 1 are $|\alpha|^2$ and $|\beta|^2$, respectively. If we receive 0, the qubit switches to state $|0\rangle$ after the measurement; if we receive 1, the qubit switches to state $|1\rangle$ after the measurement. Thus, a measurement "collapses" the superposition to one of the basis states. Mathematically, the collapse projects the state onto basis states and normalize it, i.e., scaling the vector so that the norm is 1.

A *global phase* is a complex number with unit norm, i.e., $e^{i\theta}$. A global phase does change the measurement probabilities since they are squared norms of amplitudes, which means the global phase will cancel out, e.g., $|\alpha e^{i\theta}|^2 = |\alpha|^2$.

The basis $\{|0\rangle, |1\rangle\}$ is known as the *computational basis* but we can use other bases in the measurement, e.g., the Hadamard basis $|\pm\rangle := (|0\rangle \pm |1\rangle)/\sqrt{2}$. Then, after the measurement, the state can be either in $|+\rangle$ or $|-\rangle$. The probability of getting $|+\rangle$ and $|-\rangle$ are respectively

$$|\langle +|\psi\rangle|^2 = \left|\frac{\begin{pmatrix}1 & 1\end{pmatrix}}{\sqrt{2}} \cdot \begin{pmatrix}\alpha \\ \beta\end{pmatrix}\right|^2 = \frac{|\alpha+\beta|^2}{2},\ |\langle -|\psi\rangle|^2 = \left|\frac{\begin{pmatrix}1 & -1\end{pmatrix}}{\sqrt{2}} \cdot \begin{pmatrix}\alpha \\ \beta\end{pmatrix}\right|^2 = \frac{|\alpha-\beta|^2}{2}, \quad (1.3)$$

where $\langle +|$ called '*bra* +' is the Hermitian conjugate of $|+\rangle$. In the matrix form, Hermitian conjugate is the transpose of complex conjugate, so a bra, being the conjugate of a ket, corresponds to a row vector. Thus, the product of a bra and a ket, $\langle\psi|\phi\rangle$, is a complex number, which quantifies the overlap between the two. The braket notation $\langle\psi|\phi\rangle$ is just the inner product in the vector space, $\langle\psi,\phi\rangle$. The vector form of $|+\rangle$ in the Hadamard basis is $[1,0]^T$ (where the superscript T means transpose) because $|+\rangle = 1\cdot|+\rangle + 0\cdot|-\rangle$. This is the same as the vector form of $|0\rangle$ in the computational basis. (We use brackets in the vector instead of parentheses to signify that it is associated with the Hadamard basis.) Similarly, the vector form of $|-\rangle$ in the Hadamard basis is the same with the vector form of $|1\rangle$ in the computational basis. When the basis is not specified, usually it is the computational basis and measurements are also in the computational basis.

A quantum state of a larger system consisting of n qubits is a vector with unit norm in the 2^n dimensional complex vector space. For instance, a general two-qubit state is

$$|\phi\rangle = \begin{pmatrix}\alpha \\ \beta \\ \gamma \\ \lambda\end{pmatrix} = \alpha\begin{pmatrix}1 \\ 0 \\ 0 \\ 0\end{pmatrix} + \beta\begin{pmatrix}0 \\ 1 \\ 0 \\ 0\end{pmatrix} + \gamma\begin{pmatrix}0 \\ 0 \\ 1 \\ 0\end{pmatrix} + \lambda\begin{pmatrix}0 \\ 0 \\ 0 \\ 1\end{pmatrix} = \alpha|0\rangle|0\rangle + \beta|0\rangle|1\rangle + \gamma|1\rangle|0\rangle + \lambda|1\rangle|1\rangle, \quad (1.4)$$

where the basis vectors are 4-dimensional. By writing several kets consecutively, we mean their *tensor product* and neglect the symbol $\otimes$ for convenience. A general tensor product state of two individual qubits, $|\psi_0\rangle$ and $|\psi_1\rangle$, is

$$|\psi_1\rangle|\psi_0\rangle := |\psi_1\rangle \otimes |\psi_0\rangle = \begin{pmatrix}\alpha \\ \beta\end{pmatrix} \otimes \begin{pmatrix}\alpha' \\ \beta'\end{pmatrix} = \begin{pmatrix}\alpha\begin{pmatrix}\alpha' \\ \beta'\end{pmatrix} \\ \beta\begin{pmatrix}\alpha' \\ \beta'\end{pmatrix}\end{pmatrix} = \begin{pmatrix}\alpha\alpha' \\ \alpha\beta' \\ \beta\alpha' \\ \beta\beta'\end{pmatrix}, \quad (1.5)$$

where the tensor product is computed by multiplying the operand after the $\otimes$ to each element in the operand before the $\otimes$. By combining Equation 1.2 and Equation 1.5, we can verify that our definition of two-qubit computational basis states is consistent with the vectors in Equation 1.4, e.g., $|0\rangle|0\rangle$ is indeed $(1,0,0,0)^T$.

A general two-qubit state, Equation 1.4, does not necessarily have the product form in Equation 1.5. This means that there are many two-qubit states where the qubits cannot be considered individuals. In this case, the two qubits are *entangled*.

With multi-qubit states, it is common to omit the normalization factor for convenience during computations. For instance, if we measure the first qubit in Equation 1.4 and the measurement yields 0, we typically express the resulting state as $|0\rangle(\alpha|0\rangle + \beta|1\rangle)$. Technically, the state should be normalized as $|0\rangle\left(\frac{\alpha}{\sqrt{|\alpha|^2+|\beta|^2}}|0\rangle + \frac{\beta}{\sqrt{|\alpha|^2+|\beta|^2}}|1\rangle\right)$, but this can be cumbersome, particularly if there are further computational steps. (Note that $\sqrt{|\alpha|^2+|\beta|^2}$ might not equal 1; instead, it is the total state norm, $\sqrt{|\alpha|^2+|\beta|^2+|\gamma|^2+|\lambda|^2}$, that equals 1.) Therefore, we often skip normalization in intermediate steps and apply it only at the end of the computation to simplify the expressions. This approach is especially useful when we encounter zeros, such as $\langle 1|0\rangle$, where normalization is not applicable. By ignoring normalization at first, we maintain uniformity in mathematical expressions, and the terms with zeros can be dropped in the end.

1.1.2 Quantum Gates

A quantum gate transforms an input state to an output state. Since general quantum states are complex vectors with unit norms, a general transform between states only needs to preserve the norm. Such transforms are *unitary*. The quantum state of n qubits has dimension 2^n, so the quantum gates are in the unitary group $U(2^n)$. For example, some

common single-qubit gates are X, H, S, and T. $\dagger$ means Hermitian conjugation.

$$\begin{aligned} X &= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \; Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \; H = \frac{\sqrt{2}}{2} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \\ S &= \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}, \; S^\dagger = \begin{pmatrix} 1 & 0 \\ 0 & -i \end{pmatrix}, \; T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}, \; T^\dagger = \begin{pmatrix} 1 & 0 \\ 0 & e^{-i\pi/4} \end{pmatrix}. \end{aligned} \tag{1.6}$$

Some common two-qubit gates are CZ, CNOT, and SWAP.

$$\text{CZ} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}, \; \text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \; \text{SWAP} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{1.7}$$

The CNOT (controlled-NOT) gate, operates by leaving the second qubit unchanged when the first qubit is in the $|0\rangle$ state, and by applying an X gate (which flips the amplitudes) to the second qubit when the first qubit is in the $|1\rangle$ state. Thus, CNOT is also known as CX (controlled-X). Mathematically, the effect of a CNOT can be expressed as:

$$\text{CNOT}\left[|0\rangle(\alpha'|0\rangle+\beta'|1\rangle)\right] = |0\rangle(\alpha'|0\rangle+\beta'|1\rangle), \; \text{CNOT}\left[|1\rangle(\alpha'|0\rangle+\beta'|1\rangle)\right] = |1\rangle(\beta'|0\rangle+\alpha'|1\rangle). \tag{1.8}$$

Thus, the first qubit is typically referred to as the control qubit, and the second as the target qubit. However, it is important to note that the CNOT gate does not simply "do nothing" to the control qubit, especially when the control qubit is in a superposition. This is a key difference between the quantum CNOT gate and classical gates. To illustrate, consider applying a CNOT gate to a product two-qubit state (Equation 1.5):

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha\alpha' \\ \alpha\beta' \\ \beta\alpha' \\ \beta\beta' \end{pmatrix} = \begin{pmatrix} \alpha\alpha' \\ \alpha\beta' \\ \beta\beta' \\ \beta\alpha' \end{pmatrix} \neq \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \otimes \begin{pmatrix} \beta' \\ \alpha' \end{pmatrix} = \begin{pmatrix} \alpha\beta' \\ \alpha\alpha' \\ \beta\beta' \\ \beta\alpha' \end{pmatrix}. \tag{1.9}$$

The CNOT gate flips the lower half of the state vector. As a result, the vector after applying the CNOT gate is different from the tensor product of the original control state and the flipped target state. In general, the state after a CNOT no longer has a product form, indicating that the qubits are entangled by the CNOT.

We present another interesting aspect of the CNOT gate by changing the basis. The Hadamard basis we have introduced $|\pm\rangle$ can transform to and from the computational basis $\{|0\rangle, |1\rangle\}$ by, no surprise, the Hadamard gate (H in Equation 1.6). Since $\{|0\rangle, |1\rangle\}$ are eigenvectors of Z and $|\pm\rangle$ are eigenvectors of X, they are also called Z basis and X basis, respectively. Let us consider the matrix form of the CNOT gate in the X basis:

$$(H \otimes H)^\dagger \cdot \text{CNOT} \cdot (H \otimes H) =$$
$$\frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}. \tag{1.10}$$

Recall that $|+\rangle = [1, 0]^T$ and $|-\rangle = [0, 1]^T$ in X basis. Let us apply the CNOT gate to some product states:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ 0 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ 0 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{bmatrix}, \quad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ 0 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ 1 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \end{bmatrix}. \tag{1.11}$$

We can observe that the effect of the CNOT gate in the X basis is: when the second qubit is in the state $|+\rangle$, nothing happens; when the second qubit is in the state $|-\rangle$, the first qubit is flipped from $|+\rangle$ to $|-\rangle$. Thus, if we consider the action of CNOT in the X basis, we should call the second qubit as the control qubit and the first qubit as the target qubit. The roles of control and target is reversed compared to the case in the Z basis! This further implies that we cannot interpret the CNOT gate in a classical sense.

Given all these gates, a natural question is: are they sufficient? NAND gates are sufficient for universal classical computing. It turns out that CNOT and the generic single-qubit gate below are sufficient for universal quantum computing [NC10]

$$U_3(\theta, \phi, \lambda) := \begin{pmatrix} \cos(\theta/2) & -e^{i\lambda}\sin(\theta/2) \\ e^{i\phi}\sin(\theta/2) & e^{i\lambda+i\phi}\cos(\theta/2) \end{pmatrix}. \tag{1.12}$$

We call U_3 a *parametrized gate* or a *programmable gate*. It is actually a family of gates which can be instantiated to specific gates by assigning values to the parameters, e.g., $U_3(\theta = \pi/2, \lambda = \pi, \phi = 0)$ is just H. There are other universal quantum gate sets, but we refrain from exhibiting them here since they are irrelevant for now.

Another important gate is CCNOT, or Toffoli. It flips the last qubit with X if the first two qubits are in $|11\rangle$; otherwise, it does nothing. The projector to $|11\rangle$ is $|11\rangle\langle 11|$: applying it to a quantum state $|\phi\rangle$, the $\langle 11|$ first produces the amplitude which is the "overlap" of $|\phi\rangle$ and $\langle 11|$. Then, the amplitude is multiplied to $|11\rangle$ for the component of $|11\rangle$ inside $|\phi\rangle$. The projector of the subspace orthogonal to $|11\rangle$ is $I \otimes I - |11\rangle\langle 11|$. Therefore, the matrix form of CCNOT is

$$\text{CCNOT} = (I \otimes I - |11\rangle\langle 11|) \otimes I + |11\rangle\langle 11| \otimes X = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}. \tag{1.13}$$

Similarly, the matrix form of CNOT can also be derived from the projector description: CNOT$= |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes X$ where the first qubit is the control, and the second qubit is the target. In general, any Boolean logic circuit can be first expanded to a Boolean

a)

```
g1: CX q[2],q[1]
g2: CX q[3],q[4]
g3: u1 q[3]
g4: u2 q[4]
g5: u3 q[5]
g6: CX q[1],q[5]
g7: u4 q[1]
g8: CX q[4],q[3]
g9: CX q[5],q[2]
```

b)

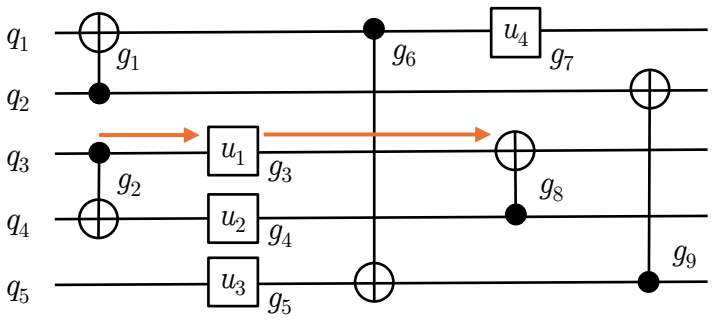


Figure 1.1: An example of unitary quantum program/circuit. **a)** A quantum program consisting of 9 gates on 5 program qubits. **b)** Circuit diagram of the quantum program. Each horizontal wire is a program qubit. Time goes from left to right. The arrow means a dependency chain (g_2, g_3, g_8).

reversible circuit, for which Toffoli is sufficient, and the Toffoli gates can be decomposed into single-qubit and two-qubit gates.

An important relation between two gates is *commutation*. If the result of first applying gate 0 followed by gate 1 is the same as the result of applying them in the reversed order, we say these two gates commute. If two gates act on disjoint qubits, they always commute. For gates acting on the same qubit, it is nontrivial to judge in general. T and CZ commute because they are both diagonal matrices; but X and H do not commute, since $XH \neq HX$.

1.1.3 Quantum Circuits

A quantum algorithm manipulates a set of qubits with quantum gates. Most notably, Shor's algorithm can factor a number N with $O(\text{polylog } N)$ quantum gates, whereas the best known classical algorithm needs $\Theta(N)$ operations [Sho94]. The readers can refer to a recent survey [DMB23] for a comprehensive list of quantum algorithms.

Quantum algorithms can be expressed in many forms, some very abstract. For our purposes, it suffices to conceptualize them as *quantum programs* written in some quantum computing instruction set. For example, in Figure 1.1a, there are several single-qubit and

two-qubit gates on five *program qubits*. A quantum program is often visualized as a *quantum circuit*, like the one in Figure 1.1b. The terms 'quantum program' and 'quantum circuit' are used interchangeably throughout this dissertation. In a circuit diagram, each program qubit is depicted as a horizontal wire, and as time progresses from left to right, gates on the wire are applied to this qubit. In our example, u_1, u_2, u_3, and u_4 are instances of U_3 (Equation 1.12), each black dot represents the control qubit of a CNOT (Equation 1.7), and each $\oplus$ symbolizes the target qubit of a CNOT.

While a quantum circuit does not explicitly schedule the gates, it does encode their relative order. In our example, there are three gates g_2, g_3, and g_8 that act sequentially on q_3. This means g_3 must follow g_2, and g_8 must follow g_3. We refer to this type of relative ordering as a *dependency*. When multiple dependencies are linked head-to-tail, they form a *dependency chain*. For instance, (g_2, g_3, g_8) is a dependency chain of length 3.

It is possible to partition a circuit into layers so that, within each layer, a qubit is involved in at most one gate. If two gates have a dependency, they cannot be placed in the same layer, but some flexibility remains. For example, g_3 could be placed in the same layer as g_5 or g_6. The minimum number of layers in a quantum circuit is referred to as its *depth*, which corresponds to the length of the longest dependency chain. The *width* of a circuit is simply the number of qubits.

Since the unitary gates belong to the unitary group, multiplying them according to the order specified in a circuit results in a unitary transformation, which we term a *unitary circuit*. However, a general quantum circuit includes components beyond unitary gates. For example, Figure 1.2a illustrates a circuit for the quantum approximate optimization algorithm (QAOA) [FGG14], where H, $U(C, \gamma)$, and $e^{-i\beta X}$ are unitary. This circuit also includes qubit initializations represented by $|0\rangle$ and measurements indicated by the meter symbols. Additionally, this algorithm features an outer layer optimization loop: based on the measurement results, a classical optimizer tunes the parameters (βs and γs) in the quantum circuit and reruns it. Such quantum circuits with tunable parameters are referred to as

variational quantum circuits.

Measurements can occur not only at the end of a circuit but also in the middle, i.e., *mid-circuit readouts*. Figure 1.2b depicts a quantum convolutional neural network (QCNN) [CCL19] circuit, where the u_is are unitary gates. In this example, mid-circuit readouts apply feed-forward gates: if the outcome of the measurement is 1, a corresponding single-qubit gate, v_i, is applied. This type of circuit is referred to as a *dynamic circuit* because the operations implemented on the quantum computer depend on measurement results known only at runtime. In contrast, the QAOA circuit parameters are determined by a classical optimizer before each execution of the circuit.

Another important family of circuits is the parity measurement circuit. When all qubits are measured, as in Figure 1.2a, the quantum state is projected onto the computational basis ($|0...0\rangle$, $|0...1\rangle$, ..., $|1...1\rangle$). However, sometimes the goal is to measure only the parity of a subset of qubits, which differs from measurement in the computational basis. For instance, measuring a *Bell state* $|\Phi^+\rangle := (|00\rangle + |11\rangle)/\sqrt{2}$ in the computational basis collapses the superposition to either $|00\rangle$ or $|11\rangle$. Conversely, measuring the parity of the two qubits does not collapse the superposition, as both $|00\rangle$ and $|11\rangle$ have even parity, preserving the state $|\Phi^+\rangle$ after the measurement. A parity measurement circuit is illustrated in Figure 1.2c. An *ancilla qubit* (the top wire) is used to collect the parity. For each qubit whose parity is to be measured, a CNOT gate is applied, controlled by this qubit and targeting the ancilla. The ancilla is initialized to $|0\rangle$ and each CNOT flips the ancilla if the controlling qubit is $|1\rangle$. An even number of flips cancels out, leaving the ancilla state as the parity of the other qubits, which is then measured at the end. This circuit measures the parity of four qubits in the Z basis. A similar circuit for measuring parity in the X basis is shown in Figure 1.2d. Several arguments can explain why this circuit measures the X parity. One elegant approach uses the insight from Equation 1.11: in the X basis, the roles of control and target in a CNOT are reversed. The initialization followed by an H gate equates to an initialization in the X basis into $|+\rangle$; similarly, the final measurement preceded by an H gate equates to a measurement

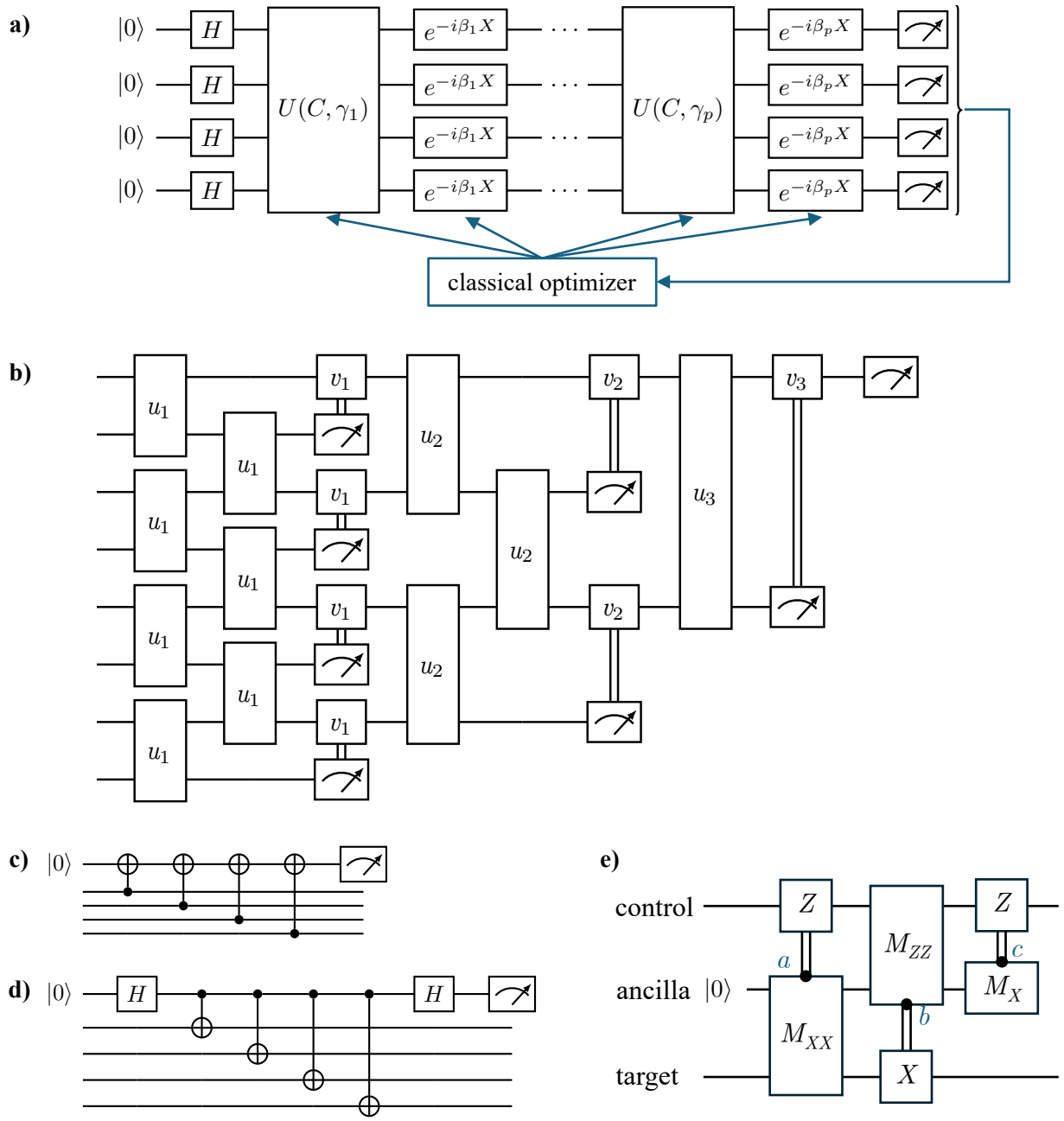


Figure 1.2: Examples of general circuits. **a)** Quantum approximate optimization algorithm circuit with initializations, parameterized unitary gates, measurements, and an outer layer classical optimizer. **b)** Quantum convolutional neural network circuit with mid-circuit readouts and feed-forward gates. **c)** Z-parity, **d)** X-parity measurement circuit. **e)** A CNOT implemented with measurements and feed-forward gates. a, b, and c are measured results.

in the X basis. In the interim, the four reversed CNOTs collect the X parity to the ancilla qubits because, in the X basis, the roles of control and target are reversed.

Figure 1.2e displays a quantum circuit equivalent to a CNOT gate, implemented using parity measurements and feed-forward single-qubit gates, which is useful because parity measurements are native to certain quantum architectures (Section 9.3.1). Demonstrating that this circuit functions as a CNOT is more elegantly done via ZX calculus, detailed in Section 9.1.4. However, we present a proof using previously introduced concepts, utilizing bit variables a, b, and c to denote the measurement results. The subscripts C, T, and A stand for the control, target, and ancilla qubits, respectively. If $a = 0$, the XX parity measurement projects the state onto the even parity subspace. The projector is expressed as $\frac{II+XX}{2}$ because for a state $|\psi\rangle$ in the even parity subspace, $XX|\psi\rangle = |\psi\rangle$, so $\frac{II+XX}{2}|\psi\rangle = |\psi\rangle$. Conversely, if $|\phi\rangle$ is in the odd parity subspace, $XX|\phi\rangle = -|\phi\rangle$, so the projector to the odd parity subspace is $\frac{II-XX}{2}$. In summary, the XX measurement is expressed as $(I_T I_A + (-1)^a X_T X_A)/2$. Feed-forward gates are conditionally activated based on the measurement results a, b, and c, such as applying Z_C^a: if the parity is even ($a = 0$), nothing is done; otherwise, a Z gate is applied the control qubit. The final measurement of the ancilla is denoted by $\langle +|_A Z_A^c$: a result of $c = 0$ leads to a $|+\rangle$ state; otherwise, it results in $|-\rangle$.

$$
\begin{aligned}
&(\langle +|_A Z_A^c Z_C^c I_T) \cdot \left(\frac{I_C I_A + (-1)^b Z_C Z_A}{2} X_T^b\right) \cdot \left(\frac{I_T I_A + (-1)^a X_T X_A}{2} Z_C^a\right) |0\rangle_A \\
&\overset{(1)}{=} \left(\langle +|_A Z_A^c Z_C^c X_T^b \;+\; (-1)^b \langle +|_A Z_A^{c+1} Z_C^{c+1} X_T^b\right) \cdot (Z_C^a I_T |0\rangle_A \;+\; (-1)^a Z_C^a X_T |1\rangle_A) \\
&\overset{(2)}{=} \langle +|Z^c|0\rangle \; Z_C^{a+c} X_T^b + (-1)^b \langle +|Z^{c+1}|0\rangle \; Z_C^{a+c+1} X_T^b \\
&\qquad + (-1)^a \langle +|Z^c|1\rangle \; Z_C^{a+c} X_T^{b+1} + (-1)^{a+b} \langle +|Z^{c+1}|1\rangle \; Z_C^{a+c+1} X_T^{b+1} \\
&\overset{(3)}{=} Z_C^{a+c} X_T^b + (-1)^b Z_C^{a+c+1} X_T^b + (-1)^{a+c} Z_C^{a+c} X_T^{b+1} + (-1)^{a+b+c+1} Z_C^{a+c+1} X_T^{b+1} \\
&\overset{(4)}{=} \begin{cases}
II + ZI + IX - ZX & \text{if } a + c = 0 \text{ and } b = 0 \\
ZI + II - ZX + IX & \text{if } a + c = 1 \text{ and } b = 0 \\
IX - ZX + II + ZI & \text{if } a + c = 0 \text{ and } b = 1 \\
ZX - IX - ZI - II & \text{if } a + c = 1 \text{ and } b = 1
\end{cases}
\end{aligned}
\tag{1.14}
$$

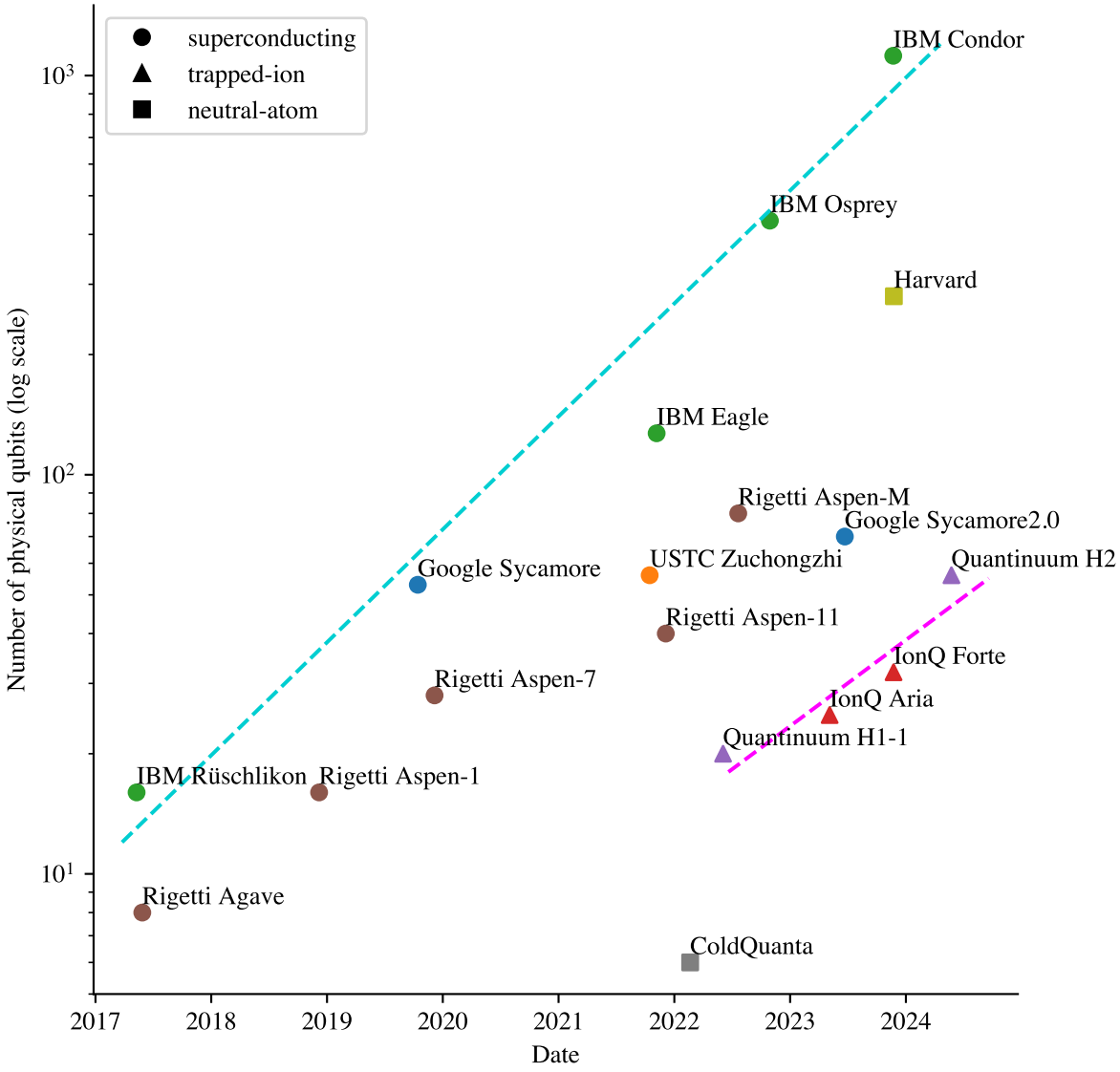
In step (1), the first two expressions are combined, and the third is multiplied by $|0\rangle_A$. Constants are ignored for simplicity. Step (2) expands and simplifies the terms, so that all the ancilla terms are combined. In step (3), we compute the ancilla terms to numbers, e.g., $\langle +|Z^c|1\rangle = (-1)^c$. This results in a sum of four terms each defined by measurement results. We observe a and c always appear as $a + c$, leading to the expansion in step (4), which presents the cases for possibilities of $a + c$ and b. Note that $II + ZI + IX - ZX = (I + Z)I + (I - Z)X = |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes X$ is a CNOT. Thus, all four possible outcomes correspond to a CNOT, demonstrating that the described circuit functions as a CNOT. The case of $a + c = b = 1$ includes a global phase of -1, which is irrelevant to the functionality.

1.2 Status of Quantum Computing Hardware

Since the discovery of quantum algorithms with potential runtime advantages over classical algorithms, significant progress has been made in scaling up quantum computing hardware to run these algorithms. In Figure 1.3, we present data showing the number of physical qubits (on a logarithmic scale) versus the release dates of quantum computers in recent years. While this collection does not capture all existing quantum computers, the data reveal that the maximum number of physical qubits has roughly grown exponentially, i.e., linear in a logarithmic scale.

The scale of a quantum computer, characterized by the number of physical qubits, is not the only metric that matters; the quality of qubits and quantum gates is also crucial. However, as long as the scale increases, we encounter growing complexity in compilation, as researchers aim to run increasingly larger quantum circuits on the hardware.

In classical computers, bits are generally encoded as voltage and can be easily "cloned" from one location on the chip to another through electric wires. Gates are physical entities that produce new bits based on the input bits. In contrast, in most quantum computing platforms, qubits themselves are physical entities. Moving qubits is much more complex than



Sources of data are as follows. Google Sycamore: [AAB19], Google Sycamore2.0: [MVM23], USTC Zuchongzhi: [WBC21], IBM Osprey: https://link.aps.org/doi/10.1103/PhysRevLett.127.180501, IBM Eagle: https://www.ibm.com/quantum/blog/127-qubit-quantum-processor-eagle, IBM Condor: https://www.ibm.com/quantum/blog/quantum-roadmap-2033, IBM Rüschlikon: https://uk.newsroom.ibm.com/2017-05-17-IBM-Builds-Its-Most-Powerful-Universal-Quantum-Computing-Processors, IonQ Aria: https://aws.amazon.com/blogs/quantum-computing/amazon-braket-launches-ionq-aria-with-built-in-error-mitigation/, IonQ Forte: https://ionq.com/news/ionqs-most-powerful-quantum-system-ionq-forte-now-available-through-the, Quantinuum H2: https://www.quantinuum.com/news/quantinuums-h-series-hits-56-physical-qubits-that-are-all-to-all-connected-and-departs-the-era-of-classical-simulation, Quantinuum H1-1: https://www.quantinuum.com/news/quantinuum-completes-hardware-upgrade-achieves-20-fully-connected-qubits, Harvard: [BLS24], ColdQuanta: [GSS22], All Rigetti data points: https://www.rigetti.com/what-we-build.

Figure 1.3: Scaling of the number of physical qubits in recent years.

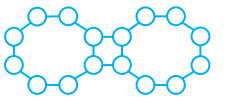
moving classical bits because quantum states cannot be cloned [WZ82, Die82]. Multi-qubit gates are not physical entities but joint operations on multiple qubits, requiring these qubits to be moved to adjacent locations.

In this section, we introduce three leading quantum computing platforms: superconducting circuits, trapped ions, and neutral atoms. This section is not intended to be an all-inclusive review of these platforms. Additionally, significant progress has also been made on other platforms, such as nuclear magnetic resonance (NMR) [LBP16], semiconductor spins [BLP23], and nitrogen-vacancy (NV) centers [PM21]. We shall not go into details on these platforms since they have not yet matched the scale or quality achieved by the first three platforms.

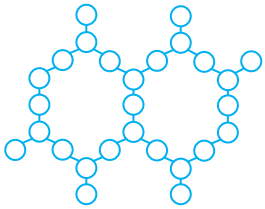
1.2.1 Superconducting Circuits

A superconducting quantum architecture can be represented by a *coupling graph*. A few examples are shown in Figure 1.4 including a recent IBM chip, Torino. In these graphs, each node represents a physical qubit, and each edge represents a *coupler* between two qubits. For instance, Torino has 133 qubits connected in a heavy-hexagon connectivity [CZY20]. The qubits and couplers are constructed from nanofabricated circuit components, resulting in a fixed coupling graph, which is why this type of architecture is referred to as *static*.

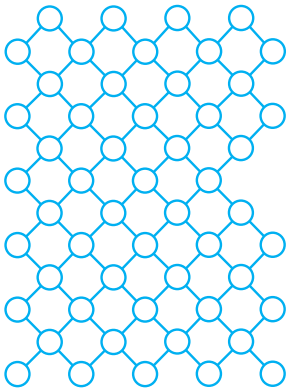
During operation, the chip must be cooled to near absolute zero temperature, in the milli-Kelvin regime, so that certain components become superconducting, allowing the quantum effects to emerge. Each qubit is connected to several control lines that extend from room temperature to the cryogenic environment. Single-qubit gates are applied by sending signals through these control lines. In current architectures, these control lines are independent. This independence allows for the simultaneous application of different single-qubit gates to any arbitrary set of qubits. However, this paradigm may need to change in the future, as it is not feasible to cool down the large number of control wires required [ALF17]. As a result, individual qubits might no longer be independently and simultaneously addressable.



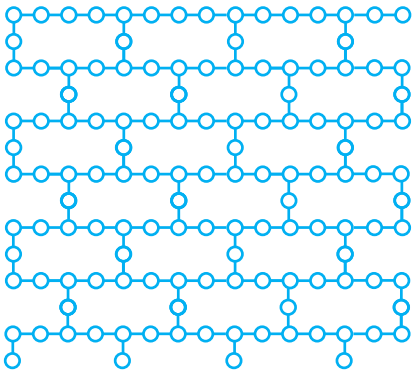
a) Rigetti Aspen-4 [CBG21]



b) IBM Falcon [JJB21]



c) Google Sycamore [AAB19]



Taken on August 15, 2024 from https://quantum.ibm.com/services/resources.

d) IBM Torino

Figure 1.4: Coupling graphs of some existing superconducting quantum architectures.

Multi-qubit gates are crucial for quantum algorithms. In superconducting circuits, only two-qubit gates are typically engineered. To perform such a gate, the two qubits must be adjacent on the coupling graph. By sending signals through the control lines of the two qubits and the coupler, a two-qubit entangling gate is executed. When two qubits are not adjacent, they cannot perform a direct two-qubit gate because there is no coupler to mediate their interaction. To resolve this issue, a SWAP gate (Equation 1.7) can be used to *logically* exchange the quantum data between two physical qubits. By performing a series of SWAP gates, two qubits can be brought adjacent on the coupling graph, allowing the desired two-qubit entangling gate to be applied.

In classical circuit design, gate fidelity is rarely a concern because fabrication processes are very mature to ensure that gates almost never produce errors under specified working conditions (one error in $\sim 10^{27}$ operations [SRA11]). However, quantum states are much more fragile, so quantum computing requires careful attention to fidelity.

There are various sources of noise in quantum computing, with decoherence being one of the most significant. Quantum data in a qubit degrade over time. The metric that

quantifies this "lifetime" of a qubit is known as the decoherence time. Calibration data for quantum computers usually provides two values, T_1 and T_2, corresponding to different types of decoherence. We use the smaller of these two as the decoherence time. Ideally, quantum algorithms should complete within a fraction of the decoherence time, as fidelity decreases rapidly when this limit is approached.

The number of gate layers that can be applied depends on the ratio of gate duration to decoherence time. For example, IBM's Torino chip has 133 qubits with a median decoherence time of 119 us.[1] The gate duration is 68 ns, allowing for over a thousand layers of gates. There are, however, several caveats to this estimation. 1) Measuring qubits at the end of a computation takes significantly longer than applying gates—for instance, Torino requires 1.56 us for measurements, though this is still small compared to the decoherence time. 2) Apart from decoherence, other types of noise listed in the next paragraph further limit the scale of circuits that can be run reliably. 3) To satisfy the connectivity constraints between qubits, additional gates like SWAPs may need to be inserted into the circuit, which further reduces the feasible circuit size.

In addition to decoherence, other types of noise affect operations on physical qubits, including single-qubit gates, two-qubit entangling gates, and SPAM (state preparation and measurement). Within SPAM, state preparation typically just involves cooling the qubits for a sufficient period of time, so the primary concern is the fidelity of measurements. Generally, for superconducting circuits, measurements have lower fidelity than two-qubit entangling gates, which in turn have lower fidelity than single-qubit gates. For instance, the mean error rates for measurements, two-qubit gates, and single-qubit gates on the Torino chip are 1.89%, 0.315%, and 0.0312%, respectively.[1] In superconducting circuits, running gates in parallel can reduce fidelity due to crosstalk between circuit components. When all two-qubit gates are executed simultaneously, the fidelity of each gate corresponds to the EPLG (error per layered gate) [MHP23]. For Torino, the EPLG is 0.662%, significantly higher than the

[1] All Torino data are taken on August 15, 2024 from https://quantum.ibm.com/services/resources.

0.315% error rate for isolated two-qubit gates.

1.2.2 Trapped Ions

Compared to superconducting circuits, trapped-ion based quantum computers generally feature fewer qubits but higher gate fidelity. For example, Quantinuum H2 boasts 56 qubits with a typical SPAM error rate of 0.15%, a two-qubit gate error rate of 0.15%, and a single-qubit gate error rate of 0.003%.[2] The H2 utilizes a fabricated chip with numerous electrodes forming a 'race track'-shaped trapping electrical potential, along which the ions can be moved [MBA23]. Unlike superconducting circuits, the quantum states of ions do not require a cryogenic environment, thus eliminating the need for cooling the chip to the milli-Kelvin regime.

The number of gate layers that can be executed is influenced not only by decoherence time and gate durations but also by the need to shuttle ions together for two-qubit gates, as these movements are relatively slow. Quantinuum has implemented a physical swap protocol that rotates two ions by 180 degrees to permute their order on the track. To simplify programming, their system presents an all-to-all connectivity to the user, while the backend automatically generates the necessary physical swaps. They define a depth-1 circuit time as the sum of 1) time required to swap the qubits from a previous permutation to the current one, 2) time to apply a layer of arbitrary single-qubit gates, and 3) time to apply a layer of two-qubit gates in the current permutation. The typical memory error per qubit at depth-1 circuit time is 0.05%, indicating the possibility of running over a thousand layers of gates with overhead associated with connectivity constraints considered.

The relatively limited scale compared to superconducting circuits is due to the fact that trapped ion chips are currently only one-dimensional because of the constraints of trapping technology. For further scaling, a promising approach is the quantum charge-coupled device

[2]All Quantinuum H2 data are taken on August 18, 2024, from the data sheet available at https://www.quantinuum.com/hardware/h2.

(QCCD) [KMW02] that lays out gate zones in 2D planes, allowing ions to shuttle between them. Given that large-scale 2D ion trap architectures are still under development, and hardware providers currently manage connectivity constraints internally, we do not focus on trapped ions in this dissertation.

1.2.3 Neutral Atoms

Recently, neutral atoms trapped in arrays of optical tweezers have emerged as a promising platform for quantum computing. These systems are readily scalable, with up to 280 qubits demonstrated at Harvard [BLS24], and further significant increases in system size are anticipated. The Harvard machine reports error rates of 0.2% for measurements, 0.5% for two-qubit gates, and 0.08% for single-qubit gates.

The architecture, known as dynamically field-programmable qubit arrays [BLS22, TBL22, TBL24, TLC24], features reconfigurable qubit connectivity that can be dynamically altered during computation, as depicted in Figure 1.5. This flexibility is achieved using a 2D grid of acousto-optic deflector (AOD) traps that can move within the plane, with the ability to adjust their spacing. Given the qubits' decoherence time of 1.5 s and assuming that each layer of gates involves five movements taking typically 200 us each [BLS22], this architecture could potentially support running over a thousand gate layers. However, gate fidelity and the variable duration of rearrangements for different layers must be considered to get an accurate estimation.

1.3 Layout Synthesis for Static Quantum Architectures

As discussed in the previous section, various quantum computing platforms exhibit distinct connectivity constraints among their qubits. The central focus of this dissertation, layout synthesis, aims to transform quantum programs to ensure they can be executed under these connectivity constraints.

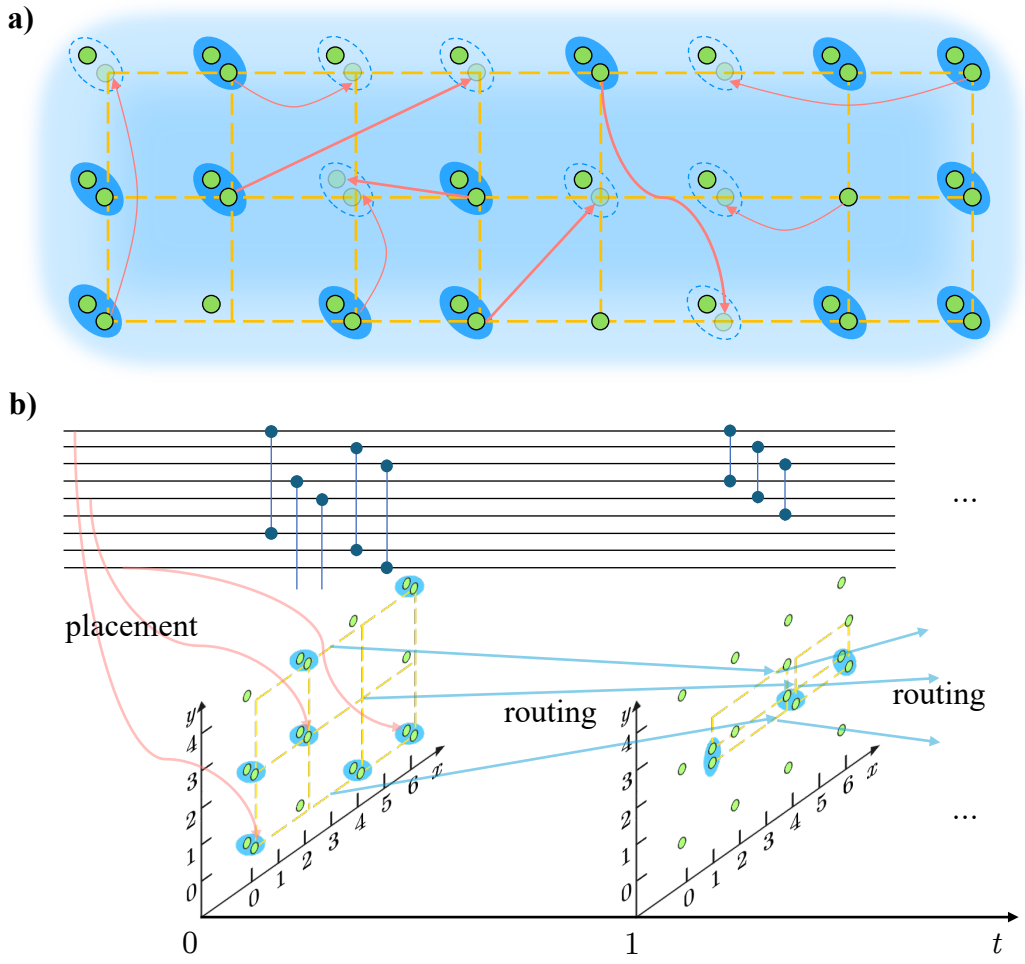


Figure 1.5: The dynamically field-programmable qubit arrays (DPQA) architecture. **a)** Non-local connectivity of DPQA. Atoms are kept in traps generated by a 2D acousto-optic deflector (AOD, dashed grid) and a spatial light modulator (SLM, all others). Entangling two-qubit gates are enabled by a Rydberg laser illuminating the plane (glow). Only when two atoms are within the Rydberg blockade range r_b can they perform an entangling gate (pairs in colored ovals). We can change the location of AOD atoms, and transfer atoms between AOD and SLM traps [BTM07] *in the middle of computation* (each arrow corresponding to some AOD reconfiguration). Through such reconfigurations, *new non-local* connectivities are established (oval dashes), i.e., different pairs of atoms can now perform entangling gates. *(This caption continues on the next page.)*

(previous page) **b)** Structure of compiled results. We discretize space by prescribing *interaction sites* shown as the proximity of integer points in the plane. The distance between sites is sufficient to suppress Rydberg interaction strengths [BLS22, EBK23] so the two-qubit entangling gates can only take place within sites. A compiler places the qubits in the quantum circuit to atoms in SLM or AOD at a specific interaction site in the beginning of execution. The time is discretized by setting *stages* when two-qubit gates are performed. After each stage, some AOD movements and atom transfers serve as routing for the gates executed at the next stage.

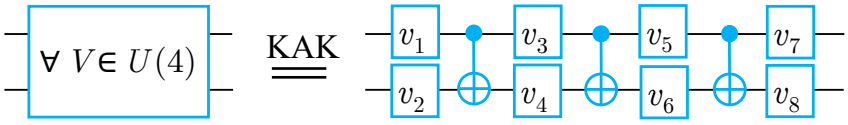


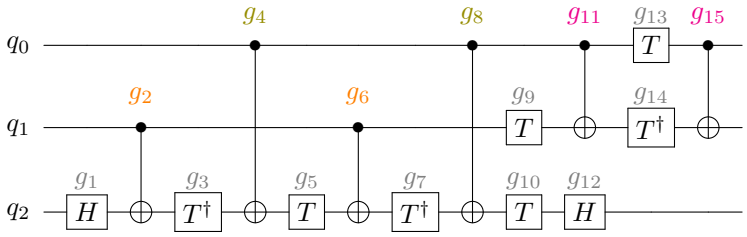
Figure 1.6: KAK decomposition. For any V that is a 4-by-4 unitary matrix, the corresponding two-qubit gate can be decomposed into 3 CNOTs and 8 single-qubit gates.

Before engaging in layout synthesis, it is essential to confirm that the quantum circuit is expressed using the native gate set of the hardware. Gates not included in this set must be decomposed into a series of executable gates from the set. Most near-term quantum algorithms, fortunately, are primarily composed of single-qubit and two-qubit gates. Moreover, canonical decompositions exist for any arbitrary two-qubit gate into single-qubit unitaries and CNOTs [VW04, PCS20], as depicted in Figure 1.6. Other important multi-qubit gates in quantum computing also have efficient decompositions [BBC95]. These decompositions are similar to *logic synthesis* in electronic design automation.

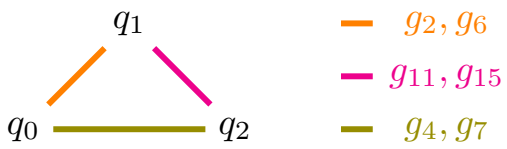
We posit that layout synthesis presents a more formidable challenge than gate decomposition. This stems from the observation that while the native gates are determined by the fundamental physics underlying the quantum computing platform, the design of coupling graphs offers greater flexibility. Indeed, the native gate set of IBM quantum computers has remained largely unchanged since the inception of their cloud-based quantum computing

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[3]; // Initiate
//     3 logical qubits
h q[2]; //g1
cnot q[1], q[2]; //g2
tdg q[2]; //g3
cnot q[0], q[2]; //g4
t q[2]; //g5
cnot q[1], q[2]; //g6
tdg q[2]; //g7
cnot q[0], q[2]; //g8
t q[1]; //g9
t q[2]; //g10
cnot q[0], q[1]; //g11
h q[2]; //g12
t q[0]; //g13
tdg q[1]; //g14
cnot q[0], q[1]; //g15
```

a) Toffoli program.



b) Toffoli circuit diagram.



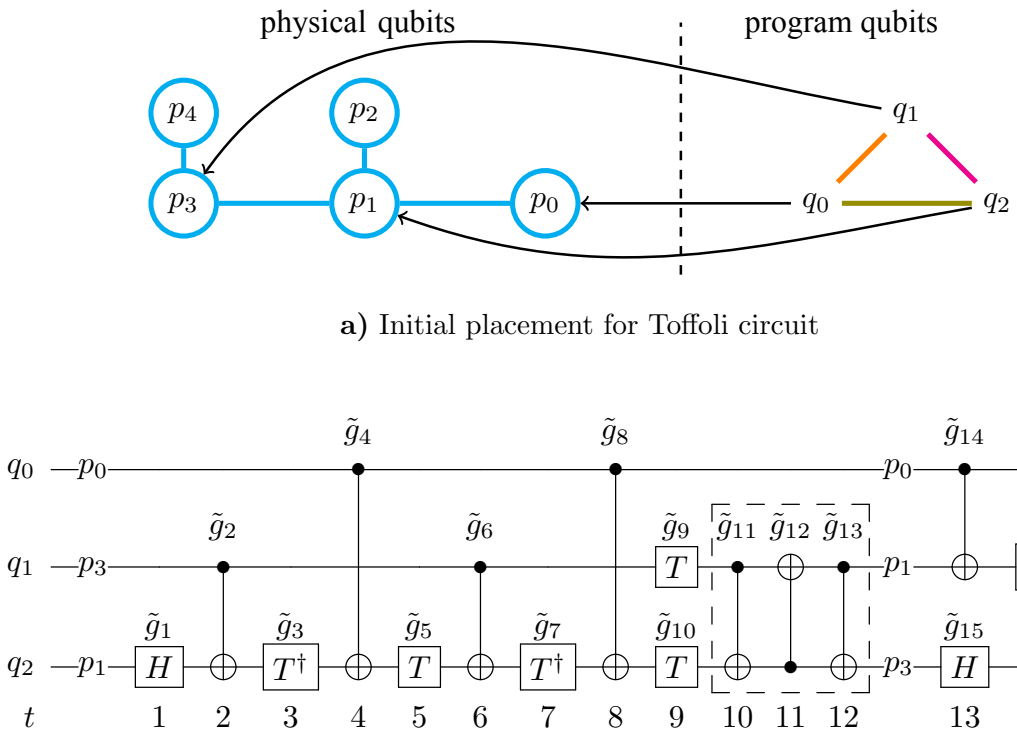
c) Qubit interaction graph of Toffoli.

Figure 1.7: Toffoli circuit. Single-qubit gates are colored gray. Identical two-qubit gates applied at different times have the same color, e.g., g_2 and g_6 are orange because they are both CNOT(q_1, q_2) but at different times.

service, as the core qubit technology has not changed drastically. However, quantum computers featuring vastly different coupling graphs have been introduced over recent years, as evidenced by the variations shown in Figure 1.4.

1.3.1 Problem Statement

A static architecture, such as superconducting circuits, has fixed connectivity between qubits, represented by a coupling graph $G = (P, E)$. For illustrative purposes, consider a small 5-qubit architecture displayed on the left of Figure 1.8a. In this graph, each vertex represents a physical qubit, and each edge allows the application of two-qubit entangling gates between connected vertices. We address the layout synthesis problem for the Toffoli circuit depicted in Figure 1.7b on the 5-qubit architecture, which comprises 9 single-qubit gates and 6 two-



a) Initial placement for Toffoli circuit

b) Scheduled Toffoli circuit

Figure 1.8: Layout synthesis for the Toffoli circuit on a 5-qubit architecture.

qubit gates. Here, the cardinality $|g| = 1$ if g is a single-qubit gate and $|g| = 2$ if it is a two-qubit gate. The notation $g \cup g'$ and $g \cap g'$ represent the sets of qubits involved in g or g' and in both g and g', respectively. For instance, in Figure 1.7b, $q_1, q_2 \in g_2$, $q_0 \in g_{13}$, and $|g_4| = |g_6| = 2$.

Layout synthesis has to produce an *initial placement*, which maps program qubits to physical qubits, a *gate scheduling*, which records the execution time of gates, and possibly some SWAP gates to adhere to connectivity constraints.

The initial placement is a map $\pi_0 : Q \to P$ to facilitate subsequent gate scheduling. By constructing a qubit interaction graph from all the two-qubit gates as shown in Figure 1.7c, we can determine if this graph can be embedded into the coupling graph so that there is no need for further transformations in gate scheduling. Typically, not all two-qubit gates can

be directly mapped to edges in the coupling graph due to topological differences, such as the absence of triangles in the coupling graph but presence in the qubit interaction graph. Nonetheless, a valid initial placement is depicted in Figure 1.8a where $\pi_0(q_0) = p_0$, $\pi_0(q_1) = p_3$, and $\pi_0(q_2) = p_1$.

Gate scheduling determines the *spacetime coordinates* (t_j, x_j) for each gate, indicating when and where gates are applied. Single-qubit gates are assigned to a physical qubit, while two-qubit gates are assigned to an edge in the coupling graph. SWAP gates are strategically placed to ensure all two-qubit gates are executable. Since only SWAP gates are inserted and all the input gates are contained in the scheduled gate list, the functionality of input circuit remains unchanged after the layout synthesis process. An example of valid, though not necessarily optimal, gate scheduling is shown in Figure 1.8b. Time coordinates for all gates are indicated at the bottom, with space coordinates derived from the mapping, e.g., $x_1 = p_1$, $x_2 = (p_3, p_1)$, and so forth. The three CNOTs $\tilde{g}_{11}$, $\tilde{g}_{12}$, and $\tilde{g}_{13}$ perform a SWAP operation, adjusting the qubit mapping to enable the execution of subsequent gates like $\tilde{g}_{14}$ and $\tilde{g}_{18}$. (We use tilde to denote that a gate is scheduled.)

Below, we provide a formal definition of layout synthesis for static quantum architectures.

Input A coupling graph $G = (P, E)$ and a list of quantum gates $g_1 ... g_M$ acting on program qubit set Q. All the input gates are in the native gate set of the architecture. Program qubits are less or equal than physical qubits, i.e., $|Q| \leq |P|$.

Output An initial mapping $\pi_0 : Q \to P$, and a scheduled quantum circuit consists of a new list of gates $\tilde{g}_1 ... \tilde{g}_{\tilde{M}}$, including SWAP gates, where each gate has a spacetime coordinate (t_j, x_j).

Constraints

Feasible two-qubit gates: all the two-qubit gates in the scheduled circuit must be on two qubits adjacent in the coupling graph. Formally, for $j = 1$ to $\tilde{M}$, if $|\tilde{g}_j| = 2$, then $x_j \in E$.

Executing all gates: all input gates should be executed. Formally, there is an injective

map $f : \{1, ..., M\} \to \{1, ..., \tilde{M}\}$ such that $g_i = \tilde{g}_{f(i)}$ for $i = 1$ to M.

Respecting dependencies: for $i, i' = 1$ to M, if $i < i'$ and $g_i \cap g_{i'} \neq \emptyset$ then $t_{f(i)} < t_{f(i')}$.

Objective The objective of layout synthesis can vary. For example, we may minimize the circuit depth T, which is the maximum time coordinate of all the scheduled gates, i.e., $T := \max_{j=1,...,\tilde{M}} t_j$. Alternatively, we may minimize the number of additional gates $\tilde{M} - M$, or the fidelity of the scheduled circuit. With fidelity as the objective, more input information may be required such as the single-qubit gate fidelity of all qubits and the two-qubit gate fidelity of all couplers.

1.3.2 NP-Hardness Results

It is natural to assess the computational complexity of layout synthesis before setting out to formulate specific solutions. We have provided a proof when the objective is to minimize circuit depth in [TC21b] which we present below. Several related results have been established, e.g., determining the minimal number of SWAP gates to insert is NP-hard by a reduction from the token swapping problem in [SSC18]. Furthermore, the NP-hardness of depth-optimal initial placement, without explicit modeling SWAP gates and scheduling, is established in [MFM08]. The NP-hardness of depth-optimal layout synthesis for QAOA circuits, given different architectural assumptions (certain edges between physical qubits are special-purposed for certain types of gates), is proven in [BKM18] through reduction from 3-SAT. Our proof is based on the Hamiltonian cycle problem, inspired by [MFM08]. However, our definition of the layout synthesis problem is different from theirs and the reduction is also different.

Theorem 1. *Depth-optimal layout synthesis is NP-hard.*

Proof. The problem of depth-optimal layout synthesis is not easier than its decision version: whether the depth of a scheduled circuit can be lower than a certain value. We reduce an arbitrary Hamiltonian cycle problem on graph $G = (P, E)$ with $|P| = N$ to a depth-decision

layout synthesis problem.

The circuit in this layout synthesis problem comprises N layers, each with a two-qubit gate $g_l = \mathrm{CNOT}(q_l, q_{(l+1) \bmod N})$, while every other qubit at each level is occupied by single-qubit gates. The coupling graph in this layout synthesis problem is directly G. Then, we ask whether the circuit can be scheduled on G with depth N as the layout synthesis problem.

Suppose the layout synthesis determines that it is indeed possible to schedule the gates with N layers, then there are no additional gates inserted because every qubit in our circuit depends on some gates in the previous layer. If a gate is inserted, some dependency chains are necessary to lengthen in the scheduling, and the depth after layout synthesis cannot be N. Because there are no new gates, the map from program to physical qubits, π, never changes, and gate $\mathrm{CNOT}(q_l, q_{l+1})$ are executed on some edge $(\pi(q_l), \pi(q_{l+1}))$ in the coupling graph, which means that $(\pi(q_1), \pi(q_2), ..., \pi(q_N), \pi(q_1))$ is a Hamiltonian cycle.

Conversely, if the layout synthesis determines that there is no solution with N layers, there is no Hamiltonian cycle in G. For contradiction, suppose there is a Hamiltonian cycle $(p_1, p_2, ..., p_N, p_1)$, then initial placement $\pi : q_i \mapsto p_i$ for $i = 1$ to N enables all gates to execute in N layers without any additional gates.

Therefore, an NP-complete problem, Hamiltonian cycle, is reducible to the depth-decision layout synthesis. This means the depth-decision layout synthesis problem is NP-complete and thus the depth-optimal layout synthesis problem is NP-hard. □

This theorem also allows us to establish the NP-hardness of gate-optimal and fidelity-optimal layout synthesis.

Corollary 1.1. *Gate-optimal layout synthesis is NP-hard.*

Proof. The decision version of gate-optimal layout synthesis involves determining if there is a layout synthesis solution with less than a certain number of gates. Specifically, it can determine whether there is a solution without any additional gates. On the family

of instances constructed in the proof for Theorem 1, a solution without additional gates is necessary to have a fixed map from program to physical qubits. Therefore, if the gate-decision layout synthesis can be determined, we can use the result to determine the Hamiltonian cycle problem. Since the latter is NP-complete, the former is NP-complete, and thus the gate-optimal layout synthesis problem is NP-hard. □

Note that this proof covers scenarios where circuit transformations in layout synthesis utilize constructs other than SWAPs, such as bridge gates [SSC18], so we term the problem gate-optimal rather than SWAP-optimal layout synthesis. Based on this result, it is straightforward to prove fidelity-optimal layout synthesis is NP-hard assuming the fidelity model is the product of all gate fidelities.

Corollary 1.2. *Fidelity-optimal layout synthesis is NP-hard.*

Proof. The decision version of fidelity-optimal layout synthesis involves determining if there is a layout synthesis solution with fidelity higher than certain constant. Specifically, it can determine whether there is a solution for the layout synthesis instances constructed in the proof of Theorem 1 with fidelity f^{N^2-N} assuming the fidelity of all types of gates is f. Since the quantum circuit has N two-qubit gates and $N^2 - 2N$ single-qubit gates, a layout synthesis solution with a fidelity of f^{N^2-N} is necessary to have no additional gates, thus a fixed map from program to physical qubits. Therefore, if fidelity-decision layout synthesis can be determined, we can use the result to determine the Hamiltonian cycle problem. Since the latter is NP-complete, the former is NP-complete, and thus the fidelity-optimal layout synthesis problem is NP-hard. □

1.3.3 Previous Work

In the most general sense, layout synthesis is generating a quantum circuit that satisfies connectivity constraints and fulfills the functionality of the input circuit. Previous works on this problem include [WIP07, MFM08, HNY11, SSP13, SSP14, KDS16, KDS18, WLD14, SSC18,

ZPW18, CSU19, IRI19, WBZ19, SSC19, CDD19, BSA19, LWD15, VDR17, VDR18, BDB18, ZW19, KHD19, TQ19, LDX19, MBJ19, MLM19, AAG19]. They can also be characterized by the optimization metrics used. It can be the additional "cost", which is usually proportional to the number of additional gates [MFM08, HNY11, SSP13, WLD14, LWD15, KDS16, KDS18, SSC18, CSU19, CDD19, IRI19, ZPW18, ZW19, SSC19, WLD14, WBZ19]; or circuit depth [WIP07, VDR17, BDB18, VDR18, BSA19]; or circuit fidelity [TQ19, AAG19, MBJ19, MLM19]; or a mix of the above [LDX19, KHD19]. We further compare these works in a few aspects.

First, these works may have some variations on the problem in mind. [WIP07, HNY11, SSP13, WLD14, LWD15, KDS16, KDS18, MBJ19] only focus on multidimensional array coupling graphs (linear array for 1D, grid for 2D, and so on). [ZW19] focuses specifically on SU(4) circuits, where all gates are two-qubit generic unitaries, and includes post-layout-synthesis optimization. [AAG19] does not focus on deriving a layout synthesis solution but adjusting the mapping after layout synthesis to improve fidelity.

In the problem statement presented above, we assume that all dependencies are respected. However, if some gates commute, changing their relative order does not alter the functionality of the circuit. The dependencies are respected in previous works listed above except [IRI19, VDR17, VDR18, BDB18]. [IRI19] considers a few basic commutation rules, but the solutions it compares to do not use dependencies to encode the relative orders between the gates. Thus, it is unclear whether the gain of [IRI19] is because of commutations or encoding with dependencies. [VDR17, VDR18, BDB18] consider the layout synthesis of QAOA circuits which are special because all the two-qubit gates used in the $U(C, \gamma)$ parts (see Figure 1.2a) commute with each other.

Since layout synthesis is NP-hard, we can also categorize previous works into heuristic ones and exact/optimal ones. The worst-case runtime of the latter scales exponentially in problem size because of the computational complexity of the problem.

In general, the heuristic works formulate layout synthesis as a search problem [SDC20,

SPS20, ZHQ21, ZPW18, MFM08, KHD19, LDX19, HNY11, SSP13, KDS16, KDS18, SSC18, CSU19, CDD19, IRI19, ZW19, SSC19]. We focus on the approaches that can target arbitrary coupling graphs because existing coupling graphs are generally not n-dimensional arrays. In the search algorithms, the state is $(\mathcal{G}, \Pi)$ where $\mathcal{G}$ contains the gates that have been considered and Π is the current qubit mapping; the action leading to another state is either changing Π by SWAPs or appending some gates into $\mathcal{G}$ if they only act on adjacent qubits under the current mapping; the cost of a Π-change is often evaluated by looking ahead a few more steps in the search tree. At the beginning of the search, $\mathcal{G}$ is just empty and there are a few different ways to find the initial mapping Π_0. [MFM08, SSC18, SDC20] use the earlier two-qubit gates to construct an interaction graph between program qubits and apply existing graph isomorphism algorithms from this graph to the coupling graph. In [SSC18], the qubit interaction graph is additionally weighted by the number of two-qubit gates between this qubit pair. [ZPW18, ZW19, SPS20, ZHQ21, CDD19, IRI19] start the search with some Π_0 and expand the search tree a few times. Then, they select the best mapping so far and use it as the real initial mapping. [LDX19] searches for the final mapping of the reversed program and uses it as Π_0 for the original program. [CSU19, SSC19] leverage existing token swapping algorithms to derive efficient sets of SWAPs to transform from Π to the next mapping.

In theory, one can derive the optimal solution by fully expanding the search tree in the heuristic search approaches, but in practice, the exact/optimal approaches formulate the qubit mapping into mathematical programming and apply a solver: [LWD15, WLD14] use PBO (pseudo Boolean optimizer), [MBJ19, WBZ19, MLM19] use satisfiability modulo theories (SMT) solvers, [SSP14, BSA19, NBG22] use integer programming (IP) solvers, and [BDB18, VDR17, VDR18] use temporal planners. To reduce runtime, many works compromise by slicing the program and only considering the next slice when inserting SWAPs [MLM19, SSP14, BSA19, NBG22].

The layout synthesis problem is still quite new to compiler and design automation communities, so the name of the problem varies. It can be placement [MFM08, WLD14], rout-

ing [CDD19], compiling quantum circuits [IRI19, ZW19, MBJ19, VDR17, VDR18, BDB18, MLM19], quantum circuit transformation [CSU19], mapping circuits to quantum architectures [ZPW18, WLD14, LWD15, WBZ19, KHD19, BSA19, LDX19], conversion [HNY11] or optimization [SSP13] of circuits in quantum architecture, realization of quantum circuits [KDS16, KDS18], or qubit allocation [SSC18, TQ19, AAG19, SSC19].

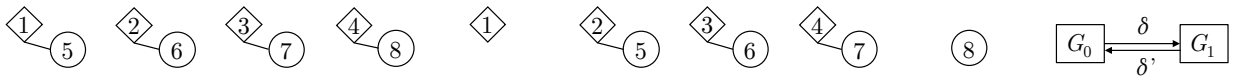
Despite all these efforts, we discovered that the heuristic approaches are still far from optimal, and the scalability of the optimal approaches can still be significantly improved, as specified in later chapters.

1.4 The Challenge of Layout Synthesis for Dynamic Quantum Architectures

In contrast to static architectures, the connectivity between qubits in dynamic architectures can be adjusted during computation. For instance, qubits can be moved using AODs in neutral atom arrays, as illustrated in Figure 1.5. Therefore, layout synthesis for dynamic architectures must also track the state of the architecture itself.

One possible approach is to extend the concept of the coupling graph to be time-dependent: at any given moment t, the coupling graph is $G_t = (P, E_t)$. Assuming that no physical qubits are created or destroyed during computation, the physical qubit set P remains unchanged, allowing us to assign fixed labels $p_1, p_2, \ldots, p_N$. However, the edge set E_t may change during circuit execution. For example, a simple dynamic architecture is shown in Figure 1.9, where diamonds represent fixed physical qubits and circles represent movable qubits that can shift left or right. The initial state of this architecture, G_0, is shown in Figure 1.9a, with the edge set $E_0 = \{(p_1, p_5), (p_2, p_6), (p_3, p_7), (p_4, p_8)\}$.

During circuit execution, a dynamic architecture can transition from one state to another, represented as traveling on a "meta-graph" $\mathcal{M} = (\mathcal{G}, \Delta)$, where each vertex itself is a possible coupling graph, i.e., a state of the architecture, and each edge δ represents a feasible state



a) G_0: the initial state. **b)** G_1: the state after shifting some qubits. **c)** Meta-graph.

Figure 1.9: A hypothetical dynamic architecture. Diamonds and circles are physical qubits.

t	τ_X	τ_{CZ}	τ_δ	τ_{CZ}	τ_X
p_1	$X[q_1]$	$CZ[q_1,\ q_2]$	δ	I	$X[q_1]$
p_2	$X[q_3]$	$CZ[q_3,\ q_4]$	δ	$CZ[q_2,\ q_3]$	$X[q_3]$
p_3	$X[q_5]$	$CZ[q_5,\ q_6]$	δ	$CZ[q_4,\ q_5]$	$X[q_5]$
p_4	$X[q_7]$	$CZ[q_7,\ q_8]$	δ	$CZ[q_6,\ q_7]$	$X[q_7]$
p_5	$X[q_2]$	$CZ[q_1,\ q_2]$	δ	$CZ[q_2,\ q_3]$	I
p_6	$X[q_4]$	$CZ[q_3,\ q_4]$	δ	$CZ[q_4,\ q_5]$	I
p_7	$X[q_6]$	$CZ[q_5,\ q_6]$	δ	$CZ[q_6,\ q_7]$	I
p_8	$X[q_8]$	$CZ[q_7,\ q_8]$	δ	I	I

Figure 1.10: Layout synthesis for the hypothetical dynamic architecture: a schedule of gates and architecture state transitions. Time goes from left to right. For every qubit, each block is either a gate or a state transition. Qubits in the same CZ gate are connected by a curve.

transition. For simplicity, we consider only one other state, G_1, as shown in Figure 1.9b. In this example, $\mathcal{M}$ has two vertices, G_0 and G_1, and two transitions: right-shift $\delta = (G_0, G_1)$ and left-shift $\delta' = (G_1, G_0)$, as illustrated in Figure 1.9c.

Suppose our quantum program consists of a layer of single-qubit gates on all qubits, followed by CZ gates on (q_1, q_2), (q_2, q_3), ..., (q_7, q_8), and concludes with some other single-qubit gates on q_1, q_3, q_5, and q_7. A valid layout synthesis result is shown in Figure 1.10, where we execute four CZ gates under architecture state G_0, perform a state transition δ, and execute three more CZ gates under state G_1. In this figure, the map from program to physical qubits is $\pi : Q \to P$ with $\pi(q_1) = p_1$, $\pi(q_2) = p_5$, $\pi(q_3) = p_2$, $\pi(q_4) = p_6$, $\pi(q_5) = p_3$, $\pi(q_6) = p_7$, $\pi(q_7) = p_4$, $\pi(q_8) = p_8$, and it remains unchanged because there are no SWAPs.

However, using the coupling graph directly as the state of the architecture has limitations,

as it only encodes connectivity and omits other critical information such as the specific location of qubits. Therefore, in layout synthesis for dynamic architectures, we still leverage the concept of a time-dependent architecture state, but the state representation is customized to the specific architecture. For neutral atom arrays, we represent the state directly with the locations of the qubits in xy-coordinates. The coupling between qubits is derived as a property of the state, depending on the distance between coordinates. Further details on this will be provided in later chapters.

A key hardware assumption of the DPQA architecture is that the Rydberg laser globally excites all qubits. An individually addressed Rydberg laser has also been demonstrated, but the two-qubit gate fidelity so far at 92.5% [GSS22] is much lower than the global approach. Previous studies on layout synthesis for neutral atom arrays have been mainly focusing on architectures with individual addressability, where the qubits can be routed logically with SWAP gates like on static architectures. Baker et al. [BLD21] covered the layout synthesis under such a hardware setting. Li et al. [LZC23] further considered the detailed durations for different gates in the scheduling. Patel et al. [PST22] proposed a method of logic resynthesis to leverage three-qubit gates available on neutral atoms. The SWAPs for routing qubits can sometimes become 'free lunch' after the resynthesis. Some works also utilize the movement capabilities on neutral atoms. Brandhofer et al. [BBP21] targeted an architecture with a more restricted kind of movement, '1D displacement'. Nottingham et al. [NPW23] and Schmid et al. [SPK23] proposed to combine the SWAP and AOD movement capabilities for routing qubits. However, the SWAPs still rely on individual addressability.

1.5 The Challenge of Layout Synthesis for Fault-Tolerant Quantum Architectures

We can try our best to reduce the overhead error incurred in layout synthesis, such as by minimizing the depth or the number of SWAPs. However, this approach does not remove

or correct any errors. Quantum algorithms of interest, e.g., [Sho99], use millions or trillions of operations, which is a much higher requirement than what state-of-the-art quantum computers, as introduced in Section 1.2, can achieve. This gap can be crossed using quantum error correction (QEC) [CRS98, DKL02, Got97]. We will introduce QEC and fault-tolerant quantum computing (FTQC) that operates on top of QEC in more detail in Chapter 9. As large-scale FTQC has yet to be realized, there are many possibilities for how such architectures might be designed, leaving the layout synthesis formulation somewhat indeterminate. We will focus on a hypothetical architecture based on surface codes, which is a promising candidate for realizing FTQC [FMM12, Lit19a, LN22].

In this code, physical qubits are laid out as illustrated in Figure 1.11a. Only nearest-neighbor connections are required. There are two types of qubits: data qubits to encode quantum data, and syndrome qubits to detect errors. In every QEC *round*, each syndrome qubit measures a *stabilizer* which is the parity of the four neighbor data qubits (or fewer if on boundaries) along the X or Z basis. With these measurements, errors can be inferred, tracked, and corrected in the classical control software.

We opt for *lattice surgery* [FG19, HFD12, Lit19a] as the FTQC scheme because it has a much lower resource overhead compared to alternative schemes like braiding [FMM12, RHG07]. In lattice surgery, logical qubits are defined on *patches* of surface code. The simplest kind of patch is a *tile*, e.g., the two tiles at the bottom of Figure 1.11b. A tile has two types of boundaries, X (red) or Z (blue), predicated by the type of 2-body stabilizer on that boundary. Tiles can be merged to larger patches. Reversely, patches can be split to smaller patches. The number of physical data qubits in a tile is $d \times d$, where d is the code distance which is the length of the longest error chain that can be caught. Since the merging and splitting in lattice surgery only concern the boundaries of tiles, we *treat a tile as the basic unit in space* in this architecture, independent from the code distance. Similarly, d QEC rounds are needed after every layer of operations, so we treat d QEC rounds as one unit of time in this architecture. When the boundaries of patches sweep through time, the

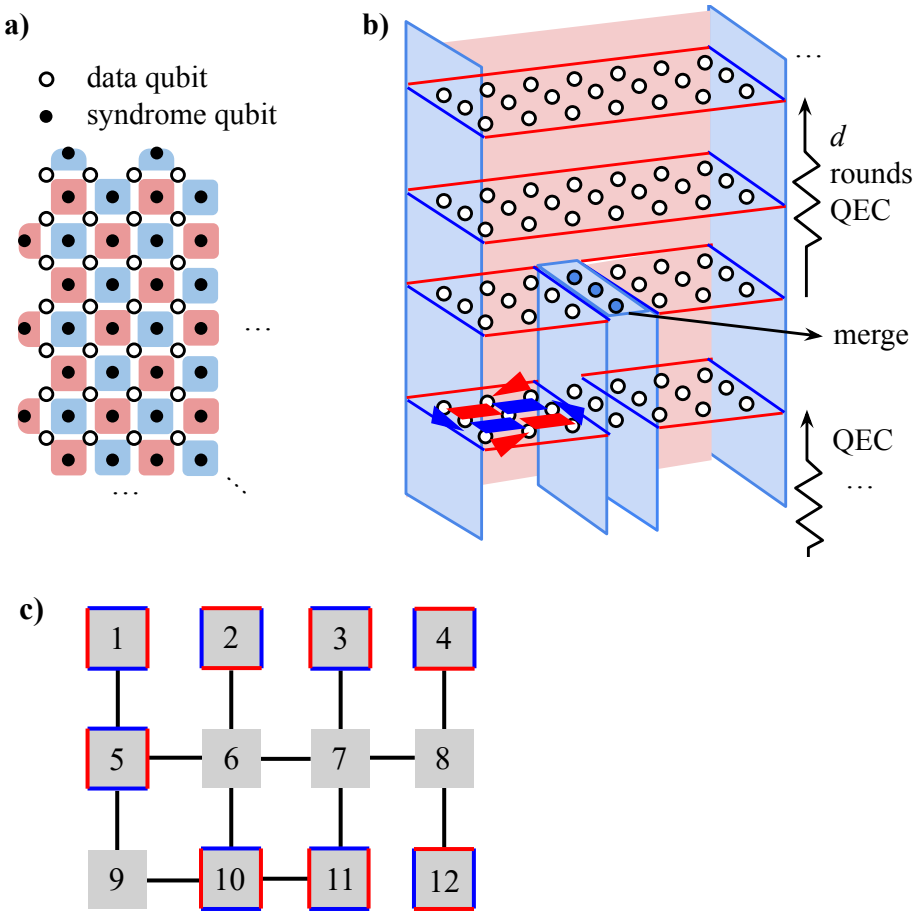


Figure 1.11: Fault-tolerant quantum computing based on surface codes. **a)** The layout of physical qubits. Red faces stand for four-body or two-body X stabilizers; blue faces are Z stabilizers. We shall use color mapping $(X,Y,Z) \mapsto$ (R,G,B) in this dissertation. **b)** An example logical operation: merging two separate tiles to a rectangular patch. The stabilizers (faces in part a), for the left tile in the beginning are drawn. The two-body stabilizers determine the type of boundaries. The X boundaries (red) are those that touch XX stabilizers (red triangles); the Z boundaries (blue) are those that touch ZZ stabilizers (blue triangles). d QEC rounds are performed after the merge. (Some rounds are omitted in the drawing.) In the 3D spacetime, the boundaries of code patches (on the qubit plane) sweeping through time produce "pipes". **c)** A sketch of the fault-tolerant architecture. Logical qubits (tiles with colored boundaries) can interact on the same type of boundary.

FTQC procedure becomes "pipes" in the 3D spacetime, exemplified by Figure 1.11b.

In Figure 1.11c, we try to extend the coupling graph notion to this architecture. The figure represents a moment in time during the computational process (which is a cross section of pipe diagrams like Figure 1.11b) where each square is a tile of surface code. We identify three key differences in the layout synthesis for this fault-tolerant architecture compared to layout synthesis problems we introduced previously.

First, the connectivity constraints between qubits are more complex than simple adjacency requirements. Assuming the architecture employs superconducting circuits with nearest-neighbor connectivity [FMM12], the qubit connectivity at the logical level is also 2D nearest-neighbor. However, not all such connections are included in Figure 1.11c. This is because the interaction between tiles is contingent not only on their adjacency but also on the orientation of their boundaries. For instance, qubits 1 and 5 have X boundaries in the vertical direction and Z boundaries in the horizontal direction, facilitating interaction between them. Conversely, qubits 1 and 2 have different orientations and cannot interact.

Second, the use of ancillas becomes crucial. Ancillas are represented by tiles without colored boundaries and are capable of interacting with all neighboring tiles, serving as intermediaries for logical operations. For example, qubits 4 and 12 can interact through ancilla 8, while qubits 3 and 10 can interact via ancillas 6 and 7. In the latter case, if this is a layout synthesis for static architectures, qubits 6 and 10 need to be swapped, and so do qubits 3 and 7, to enable interaction between qubits 3 and 10. However, fault-tolerant architectures allow for leveraging paths consisting of multiple ancillas to facilitate long-range interactions, where the length of the path does not impact the logical circuit depth, which remains at one unit of time [Lit19a]. This capability significantly alters the dynamics of layout synthesis. Nonetheless, there are specific restrictions on interactions through ancilla paths: the endpoints of the path must have matching boundaries. For example, qubits 4 and 10 cannot interact through the path 6-7-8 because this path connects a Z boundary (top of 10) to an X boundary (bottom of 4). Furthermore, multiple paths cannot intersect; for instance,

the paths 10-6-7-3 and 4-8-12 can be activated simultaneously, but the paths 2-6-7-8-12 and 3-7-11 cannot, as these two paths would intersect at ancilla 7.

Third, the connection between logic and layout is much stronger. While unitary gates are still useful for understanding algorithms in fault-tolerant quantum computing, the native operations may not be unitary. For example, parity measurements can be implemented with lattice surgery and thus are native to this architecture, whereas the CNOT is not native. Consequently, logic synthesis should generate circuits using parity measurements and other native operations, such as decomposing the CNOT as shown in Figure 1.2e and proved in Section 1.1.3. However, in generating such a circuit, the layout must also be considered, particularly the orientation issue mentioned earlier. In the example of Figure 1.2e, the ancilla must interact with the control qubit on a Z boundary and with the target qubit on an X boundary. If we want to apply a CNOT with qubit 5 as control and qubit 10 as target, the ancilla can be 9 but not 6, because 9 exposes the correct boundaries (Z boundary with 5 and X boundary with 10), whereas 6 does not. If we wish to use 6 as the ancilla, some different decomposition of the CNOT is required. Therefore, logic and layout synthesis are tightly interconnected for this fault-tolerant architecture.

CHAPTER 2

Overview

In Chapter 1, we introduced three distinct layout synthesis problems for static, dynamic, and fault-tolerant architectures, respectively. This chapter provides an overview of our contributions to these problems, with detailed discussions to follow in subsequent chapters, as summarized by Figure 2.1. Based on the nature of these problems and the historical research context, we adopted three high-level methodologies.

For static architectures, which have been the subject of research for over a decade as discussed in Section 1.3.3, we employed a measure-then-improve methodology. Initially, we assessed how close existing solutions at the time were to being optimal. Upon discovering significant gaps in optimality, we developed new approaches aimed at closing these gaps.

The topic of layout synthesis for dynamic architectures is relatively nascent, with limited prior work. Consequently, our methodology here focused on the precise formulation of the layout synthesis problem. A clear problem formulation generally facilitates the direct application of modern automated reasoning tools that can solve limited-size layout synthesis problems. Insights from our initial investigations also led us to refine the problem formulation, enabling efficient and near-optimal solutions.

With fault-tolerant architecture, Section 1.5 has pointed out a strong interplay between logic and layout. Accordingly, our methodology involved breaking the abstraction layers of logic and layout, addressing them as a unified task. While it may be necessary to separate these tasks in the future, our approach has proven effective for hyper-optimizing limited-scale subroutines, which can be integrated into larger computational procedures.

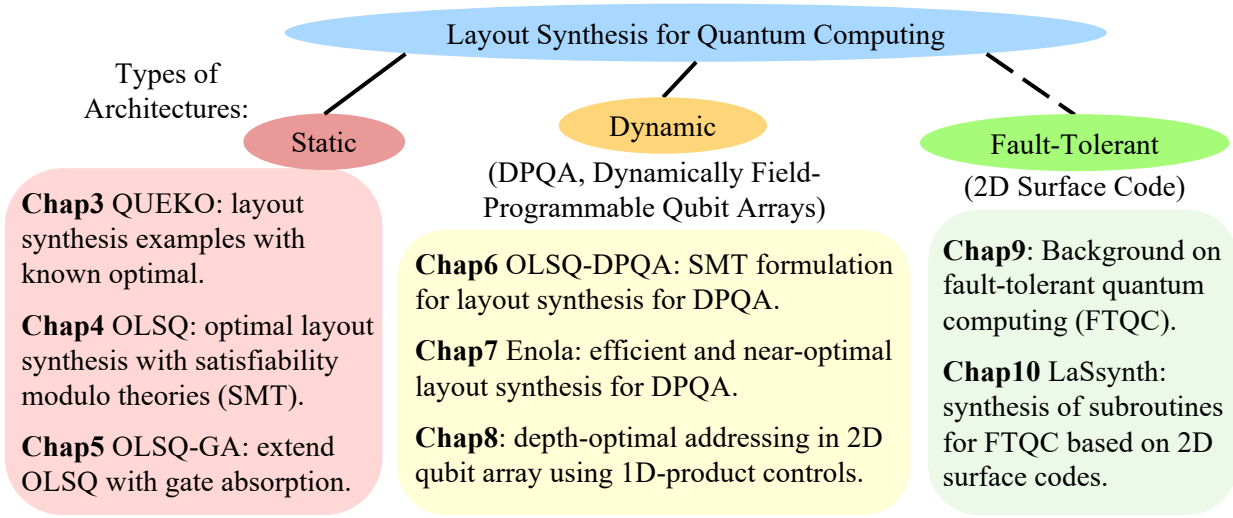
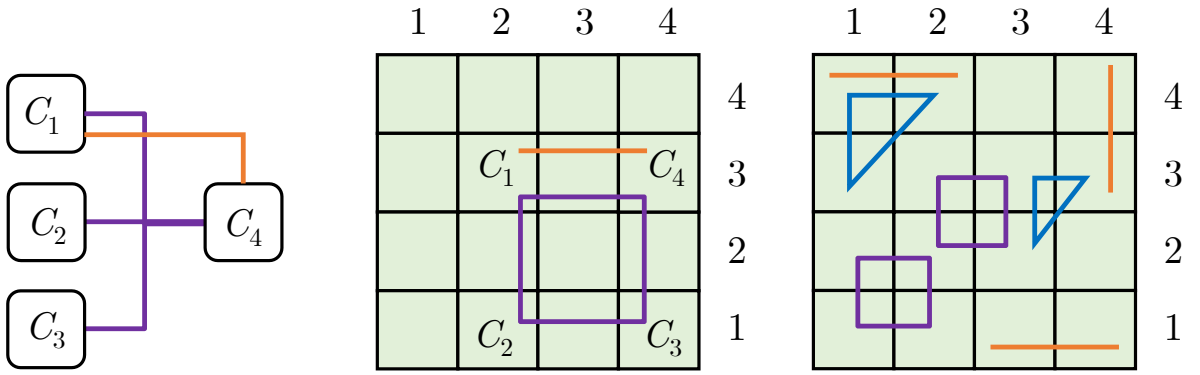


Figure 2.1: Roadmap of this dissertation. The dashes connecting 'Fault-Tolerant' signify that our contribution in this branch synthesizes logic and layout together. In comparison, on the other two branches, the focus of our contributions is layout.

2.1 The Measure-Improve Process in Layout Synthesis for Static Quantum Architectures

The layout synthesis problem is representative of many problems in electronic design automation or, broadly, in computer science: the complexity is NP-hard, and they can be formulated into some kind of mathematical programming and solved with exponential runtime. To solve large instances of these problems, one approach is to accelerate the solver, often in a domain-specific way. Another approach is to develop heuristic methods that run faster but are not optimal. How do we evaluate these heuristics? A common way is using a set of representative applications as the benchmark and comparing the results by different heuristics. However, because of the complexity of the problem, we do not know the optimal result of these benchmarks, so we do not know how much room of improvement there is. If, after substantial research, the improvements are diminishing, the community may be in a dilemma: is it possible that the current heuristics are very close to optimal and further



a) Netlist, input of the placement problem. Each net connects some pins on modules $C_i, i \in [4]$. There is a 2-pin net (orange) between C_1 and C_4, and a 4-pin net (purple) connecting all modules.

b) A placement solution. The modules are placed to cells in the chip area. This solution is not ideal since the modules are placed farther away than necessary, so longer wires are needed by the nets.

c) Construction of PEKO, placement example with known optimal [CCR04]. The nets are all shortest possible, so the shortest wire length of the whole netlist is just the sum of each one.

Figure 2.2: The placement problem in classical circuit design.

research will produce diminishing returns; or is there still significant room requiring fresh ideas and more efforts? We cannot be certain about both possibilities since deriving optimal solution for large instances takes astronomical time. In this case, it would be helpful if there were benchmarks with known optimal solution and the size of these benchmarks should be large enough to imitate real applications. With such benchmarks, we can measure the sub-optimality of the heuristics and improve them if there is still significant room.

One such benchmark set in classical circuit design is PEKO [CCR04], *placement examples with known optimal*. Before placement, the circuit is represented as connected *modules* shown as C_i, $i \in [4]$, in Figure 2.2a. The input of the problem is a *netlist* where each *net* connects two or more pins on different modules. In our example, there is a 2-pin net connecting C_1 and C_4, and a 4-pin net connecting all the modules. After placing the modules on the chip area, manufacturers need to implement the nets with wires, as shown in Figure 2.2b. This

example is not ideal since the total wirelength can be reduced if we place the modules closer together, e.g., by putting the modules at the four cells in the bottom right corner. Despite the fact that placement is known to be NP-hard [SB80], Chang et al. [CCR04] presented a way of constructing placement examples with known optimal wirelength from locally optimal nets, as illustrated in Figure 2.2c. That is, one follows the net size distribution specification. For each net of size r, one connects pins from $\lceil\sqrt{r}\rceil \times \lceil\sqrt{r}\rceil$ adjacent modules. Since all the nets are among adjacent modules, the total wirelength cannot be reduced. Thus, we know the optimal wirelengths of PEKO benchmarks by construction. The PEKO benchmarks were used to measure optimality of leading placers at that time and showed 2x optimality gap, which spurred the community to invest more efforts into the placement problem. This led to a wirelength reduction equivalent to two generations of hardware scaling in Moore's law [Sem07], but from better placement algorithms.

2.1.1 Measuring Optimality with QUEKO

To measure the optimality of existing layout synthesis solutions for static quantum architectures, we developed QUEKO [TC21b] – quantum mapping examples with known optimal, inspired by PEKO. We used QUEKO to evaluate the optimality of layout synthesis tools at the time, including Cirq from Google [Dev21], Qiskit from IBM [AAA21], t|ket⟩ from Quantinuum (Cambridge Quantum Computing at the time) [SDC20], and leading academic work at the time [ZPW18]. To our surprise, despite over a decade of research and development by academia and industry on compilation and synthesis for quantum circuits, we were still able to demonstrate large optimality gaps: 1.5-12x on average on a smaller device and 5-45x on average on a larger device at the time. This suggests substantial room for improvement of the efficiency of quantum computer by better layout synthesis tools. For more detailed information about QUEKO, please refer to Chapter 3.

2.1.2 Closing the Optimality Gap with OLSQ

After the optimality gaps are revealed by QUEKO, we set out to close the gaps with OLSQ – optimal layout synthesis for quantum computing [TC20] where we encode the layout synthesis problem for static architectures to satisfiability modulo theories (SMT) and invoke an SMT solver [dB08] for optimal solutions.

SMT is an extension of satisfiability (SAT) that accommodates a broader range of variable types beyond binary variables, as well as diverse types of constraints that go beyond the conjunctive normal form. We can encapsulate the variable definitions and constraints expressed with these variables in an SMT *model*. When provided with a model, an SMT solver can check whether it is satisfiable. If so, the solver returns the variable assignments which completely encode the layout synthesis solution. If the model is not satisfiable, some of our bounds, e.g., depth, are too small for valid variable assignments that will satisfy all the constraints, so we need to adjust these bounds.

Compared to the previous optimal approaches, the main contribution of OLSQ is reducing the number of variables. In [WBZ19], there is a binary variable $x_{\Pi,t}$ at each time step t for every possible map from program to physical qubits $\Pi : Q \to P$. Note that there are exponentially many possible maps with respect to the number of qubits, so there are exponentially many variables. In comparison, at each time step, OLSQ only has linearly many variables in the number of qubits, which is an improvement over previous works.

By slightly changing our formulation, we arrive at an approximate synthesizer that is even more efficient, TB-OLSQ, where 'TB' stands for 'transition-based'. TB-OLSQ outperformed some leading heuristic approaches at the time [SDC20, MLM19], in terms of additional gate cost, by up to 100%, and also fidelity, by up to 10x, on a comprehensive set of benchmark programs and architectures. For a family of QAOA circuits, we further adjust TB-OLSQ by taking commutation into consideration, achieving up to 75% reduction in depth and up to 65% reduction in additional cost compared to the tool used in a leading QAOA study at the

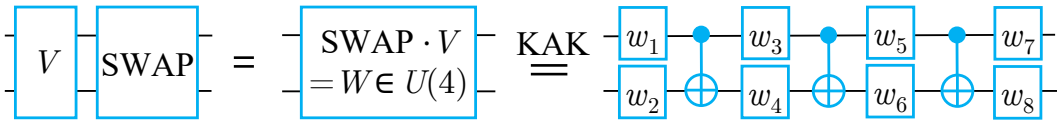


Figure 2.3: SWAP absorption. In the first step, we compute the matrix product, W, of the SWAP and the gate V; in the second step, we apply the KAK decomposition to W.

time [HSN21]. For more detailed information about OLSQ, please refer to Chapter 4.

2.1.3 Exploring Larger Solution Space with OLSQ-GA

Given domain knowledge, we can further improve solution quality. One example is the technique of gate absorption, resulting in OLSQ-GA [TC21a] where we combine layout synthesis with part of logic synthesis, i.e., implementing programmable two-qubit gates using the KAK decomposition as shown in Figure 1.6. If a SWAP gate is directly after another two-qubit gate, e.g., in a QAOA circuit or other important circuits for chemistry [KMW18] or machine learning [CCL19], we can combine these two gates by computing the matrix product of them and synthesis this product, as illustrated by Figure 2.3. The cost of implementing the gate induced by the product is much less than the cost of implementing the original two gates separately. On a set of QAOA benchmarks, OLSQ-GA reduces depth by up to 50.0% and SWAP count by 100% compared to TB-OLSQ, which translates to 55.9% fidelity improvement. For more detailed information about OLSQ-GA, please refer to Chapter 5.

2.2 Inventing and Refining Formulations in Layout Synthesis for Emerging Dynamic Quantum Architectures

Compared to static architectures, dynamic architectures are still nascent. The dynamically field-programmable qubit arrays (DPQA) based on neutral atoms were demonstrated only in 2022 [BLS22]. For such emerging architectures, the initial step in addressing the lay-

out synthesis problem is to develop a sound formulation. Essentially, formulation involves translating the problem as understood from hardware developers into mathematical variables and constraints. Once the problem is formulated, it may sometimes be straightforward to develop algorithms that directly solve the mathematical representation. However, it is more common to rely on existing tools as "oracles" to solve the mathematical problem. For instance, we might use SMT solvers, as previously mentioned, or solvers of graph theory problems. Bridging the gap between the physical realities of the hardware and the abstract mathematical models can often be complex, which is the value of a few of our contributions.

A story from *The History of Herodotus* (Book 1, lines 53 and 91) about the ancient king Croesus illustrates the complexities involved in dealing with oracles. Croesus, pondering whether to wage a war against the Persians, sent messengers to the temples to dedicate offerings and inquire of the Oracles. The Oracles cryptically replied that 'if Croesus marched against the Persians, he would destroy a great empire'. Interpreting this as a favorable prophecy, Croesus mobilized an army and initiated the war. Unfortunately, the campaign failed, and Croesus was captured by the Persian ruler, Cyrus. When Croesus later sent messengers again to confront the prophetess, he was informed that he should have further inquired whether the empire that the oracle referred to is his own empire or that of Cyrus, so his failure to seek clarification was the reason for his downfall. While the mathematical solvers we use are not as inscrutable as divine oracles, the task of formulating problems for these solvers presents similar challenges. Formulation requires setting constraints that exclude all scenarios which invalidate a solution. This task can be demanding because, when gaining insights into a problem, our focus tends to be on what constitutes a valid or good solution, rather than on avoiding *all* possible invalid solutions. For solvers, however, the principle is akin to a legal maxim: everything which is not explicitly forbidden is allowed. This characteristic is both a strength, as it often surpasses human intuition in solution space exploration, and a source of difficulty in ensuring that problem formulations are precise and comprehensive.

2.2.1 Axiomatic Formulation of the Dynamically Field-Programmable Qubit Arrays Architecture and its Layout Synthesis with OLSQ-DPQA

Before embarking on the layout synthesis for the newly developed architecture, dynamically field-programmable qubit arrays, it is crucial to first accurately characterize the architecture itself. Typically, this involves writing several paragraphs that describe the architectural principles and discuss properties relevant to layout synthesis. However, this approach can lead to confusion if too many details are included. To introduce structure and clarity into this description, adopting an axiomatic approach can be particularly effective.

Centuries ago, *René Descartes*, while contemplating philosophy in his chamber, realized that existing metaphysics at his time was a web of dubious arguments and ideas. To establish a solid foundation for metaphysics, he introduced a single axiom: *Cogito, ergo sum* (I think, therefore I am). Similarly, *Spinoza*, in his work *Ethics, Demonstrated in Geometrical Order*, applied this methodological rigor to ethics—a field seemingly unrelated to geometry. By presenting a small set of definitions and axioms, Spinoza employed logical deductions, akin to those in Euclidean geometry, to derive ethical propositions.

While the complexities of our new architecture are certainly less than those of ethics, we endeavor to adopt a similar axiomatic approach. By defining axioms which are implications of the physics of the architecture, we ensure that any layout synthesis formulation adhering to these axioms is considered correct. Just like many of Spinoza's ethical axioms are subject to debate today, our implications may also become outdated in the future. Nonetheless, the structured approach still provides a means to trace the deductions and examine the consequences for layout synthesis should any axiom change. The DPQA implications are detailed further in Section 6.1. For an overview, however, we will use natural language to describe the architecture in this section, providing readers with a glimpse of what the layout synthesis problem entails.

In the DPQA architecture, qubits are captured in two kinds of traps. A spatial light

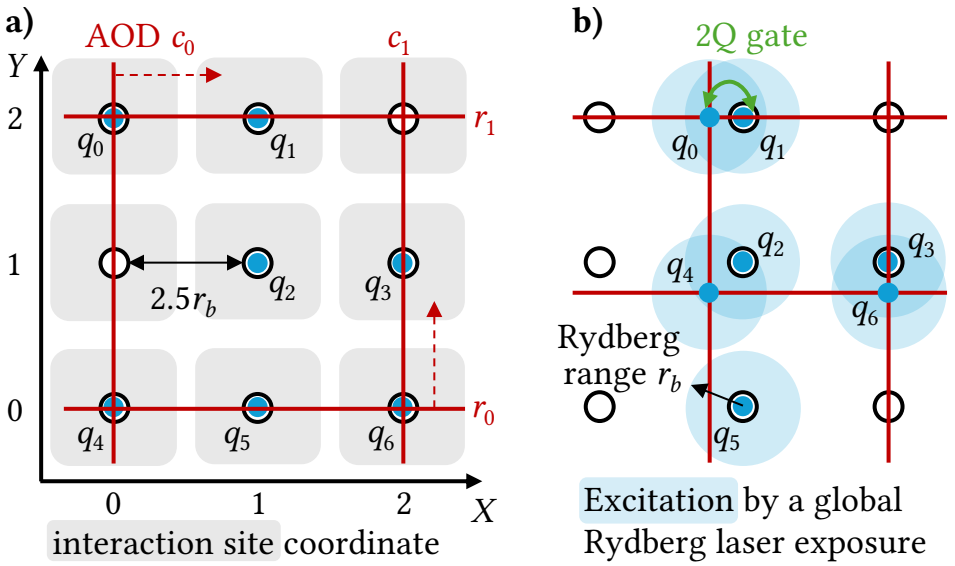


Figure 2.4: Operations in dynamically field-programmable qubit arrays. **a)** Qubits (blue dots) can transfer between SLM traps (circles) and AOD traps (intersections of red lines). AOD rows and columns can move while preserving their relative order. **b)** A global Rydberg laser excites all qubits. A two-qubit gate is applied if two qubits are in the Rydberg range.

modulator (SLM) generates an array of *static* traps, as indicated by the 3-by-3 circle array in Figure 2.4a. Seven of these traps are occupied by qubits. A 2D acousto-optic deflector (AOD) generates *mobile* traps that can travel in the plane. The AOD traps are intersections of a set of rows and columns. In our example, there are two rows (r_0 and r_1) and two columns (c_0 and c_1). When we align the AOD traps with SLM traps and ramp up the AOD intensity, qubits are *transferred* from the SLM to the AOD. In Figure 2.4a, three qubits (q_0, q_4, and q_6) get transferred to the AOD. Then, the AOD row r_0 shifts upward while the AOD column c_0 shifts to the right, taking the qubits in the AOD along. This movement yields the new configuration shown in Figure 2.4b. At this point, if we reverse the movement and wind down the AOD, the three qubits would be transferred back to the SLM. A major constraint of the movements is that the order of AOD columns cannot change, e.g., c_0 cannot move past c_1 to the right side. The order of rows also cannot change. An order violation may cause the qubits in the AOD to collide and be lost.

A global Rydberg laser, which *excites all qubits* to potentially entangle with each other, induces the two-qubit operation in DPQA. The range of this interaction is named the *Rydberg range*, r_b, illustrated by the half-transparent blue spheres in Figure 2.4b. If two qubits are within r_b of each other, a two-qubit gate is applied. In our example, three gates are applied: (q_0, q_1), (q_2, q_4), and (q_3, q_6). We call these parallel gates induced by the Rydberg laser a Rydberg *stage* in the circuit execution. Across the stages, qubits can be rearranged to different *interaction sites* to interact with different qubits. These sites are represented by the gray regions in Figure 2.4a. They center at integer points in the coordinate system and are separated sufficiently by $2.5r_b$ so that two-qubit interactions can only happen between qubits at the same site. Note that, even if a qubit is alone during a Rydberg stage so that it does not go through a gate, such as q_5 in Figure 2.4b, it still gets excited by the Rydberg laser and accumulates error. Therefore, we should minimize the number of stages to reduce these side effect errors.

With time discretized to Rydberg stages and space discretized to interaction sites, in OLSQ-DPQA [TBL22, TBL24], we formulate the layout synthesis for DPQA as SMT models, which can be solved by existing solvers optimally in terms of the number of stages. For a set of benchmark circuits generated by random graphs with complex connectivities, OLSQ-DPQA reduces the number of two-qubit entangling gates on small problem instances by 1.7x compared to near-optimal compilation results on a static 2D grid architecture. To further improve scalability and practicality of the method, we introduced a greedy heuristic inspired by the iterative peeling approach in classical integrated circuit routing [CHS93]. Using a hybrid approach that combined the greedy and optimal methods, we demonstrated that our DPQA-based compiled circuits featured reduced overhead scaling compared to the static 2D grid architecture, resulting in 5.1x fewer two-qubit gates for 90 qubit quantum circuits. For more detailed information about OLSQ-DPQA, please refer to Chapter 6.

2.2.2 Refined Formulation Enabling Efficient and Near-Optimal Layout Synthesis for Dynamically Field-Programmable Qubit Arrays with Enola

While the SMT formulation in OLSQ-DPQA adheres to all the physical implications, considered as the axioms of the architecture, it misses certain opportunities by combining a Rydberg stage with only one movement. This is more of a design compromise than a flaw of the formulation. If more movements are considered, they must be treated as separate stages, necessitating the introduction of a binary variable for each stage to determine whether the Rydberg laser should be activated. Some stages are purely for rearrangement, requiring the deactivation of this variable to prevent unwanted Rydberg interactions. Incorporating these activation variables and rearrangement stages into the SMT model is feasible but will further lengthen the runtime of the SMT solver, so we opt to not do so in OLSQ-DPQA. This limitation leads to sub-optimal fidelity in layout synthesis results.

Motivated by this observation, we refined the DPQA layout synthesis formulation to a few tasks: *scheduling* which assigns two-qubit gates to Rydberg stages, *placement* which maps the qubits to sites at different stages, and *routing* which transfers and moves qubits between stages. As a result, we developed Enola (efficient and near-optimal layout synthesizer for atom arrays) [TLC24] that has both better quality and scalability than OLSQ-DPQA. The quality improvements of Enola are mainly due to the reduction of Rydberg stages. Specifically, we can model two-qubit gates as edges in a qubit interaction graph so that the scheduling task becomes an edge coloring in the graph. Suppose the optimal number of Rydberg stages is S_{opt}. Leveraging an efficient and provably near-optimal edge-coloring algorithm [MG92], Enola manages to schedule the gates to S_{opt} or $S_{\text{opt}}+1$ stages. The placement problem is solved by simulated annealing to reduce the qubit traveling distance, and the routing solution is generated by solving independent sets to avoid AOD order violations. For the 90-qubit QAOA 3-regular MaxCut benchmarks, Enola produces 3.7x fewer stages and improves the overall fidelity by 5.9x compared to OLSQ-DPQA. Furthermore, Enola can handle much larger circuits because it consists of scalable algorithms. We demonstrate

compiling circuits with up to 10,000 qubits in 30 minutes, compared to OLSQ-DPQA's 90 qubits in a day. For more detailed information about Enola, please refer to Chapter 7.

2.2.3 Formulation of Addressing 2D Qubit Arrays with 1D-Product Controls

In the two DPQA layout synthesis contributions discussed earlier, we did not specifically address the issue of picking up atoms, as our fidelity analysis (Figure 7.1) indicated that this was not a bottleneck at the moment. However, we now shift our focus to this issue, which is closely related to the 1D-product nature of Acousto-Optic Deflectors (AODs).

In Chapter 8, we consider the problem of achieving depth-optimal AOD addressing, which we formulate as *exact binary matrix factorization* (EBMF) [TPC24]. We present an SMT formulation for this problem and an effective heuristic dubbed *row packing*. The combined algorithm, SAP (SMT and packing), finds high-quality heuristic solutions quickly and then iteratively approaches the optimal solution.

The NP-hardness of EBMF indicates the hardness of optimally picking up atoms in DPQA. Based on this result, we are able to prove the NP-hardness of routing in Section 8.5, one of the three tasks in the formulation of Enola. However, it should be noted that picking up atoms is not the bottleneck of fidelity, so a high-performant heuristic approach like the row packing algorithm suffices.

2.3 Synthesis of Subroutines for Surface-Code Fault-Tolerant Quantum Architectures with LaSsynth

As discussed in Section 1.5, the logic and layout synthesis are closely connected with each other when we use lattice surgery to interact multiple qubits in a surface-code fault-tolerant architecture. Thus, we break the separation between logic and layout and formulate a wholistic synthesis problem for logic and layout. In contrast to previous works that aims

to be end-to-end compilers for fault-tolerant quantum computing (FTQC), the focus of our contribution is hyper-optimizing certain limited-size logical blocks we named lattice-surgery subroutines (LaS). Given the frequent use of these blocks, it becomes crucial to optimize their design in order to minimize the overall spacetime volume of FTQC. We define the variables to represent LaS and the constraints on these variables. Leveraging this formulation, we develop a synthesizer for LaS, LaSsynth [TNG24], that encodes a LaS construction problem into a SAT instance, subsequently querying SAT solvers for a solution. Starting from a baseline design, we can gradually invoke the solver with shrinking spacetime volume to derive more compact designs. Due to our foundational formulation and the use of SAT solvers, LaSsynth can exhaustively explore the design space, yielding optimal designs in volume. For example, it achieves 8% and 18% volume reduction respectively over two states-of-the-art human designs for the 15-to-1 T-factory, a bottleneck in FTQC. Since T-factories can occupy 30% of the spacetime volume in Shor's algorithm [GE21], our result potentially reduces resource requirement of Shor's algorithm by 30%×18%=5.4%. For more detailed information about LaSsynth, please refer to Chapter 10.

CHAPTER 3

QUEKO: Optimality Benchmarks for Layout Synthesis for Static Quantum Architectures

In this chapter, we provide the construction of QUEKO benchmarks and the optimality experiments of existing layout synthesis tools at the time [TC21b].

3.1 QUEKO Benchmarks Construction

QUEKO is inspired by PEKO [CCR04], placement examples with known optimal. Placement is a crucial step in classical integrated circuit design, where modules are placed on a chip with the objective of minimizing total wirelength. Although this problem is NP-hard, the PEKO algorithm is able to generate benchmarks with known optimal solutions.

Similarly, for a generic input quantum circuit and a generic coupling graph, layout synthesis with optimal depth is NP-hard, which was proved in Section 1.3.2. However, it is feasible to construct some benchmarks with known optimal solutions. Given a target coupling graph G and a target depth T, we can construct a depth-optimal circuit. Then, by re-labelling the qubits, we derive a QUEKO benchmark.

Additionally, QUEKO can be customized for a given feature: gate density vector (d_1, d_2). The two components intuitively stand for the densities of single-qubit gates and two-qubit gates in the whole circuit. Suppose a circuit has n program qubits, M_1 single-qubit gates, M_2 two-qubit gates, and the longest dependency chain of length l, then $d_1 = M_1/(n \cdot l)$ and $d_2 = 2M_2/(n \cdot l)$. For example, in Figure 1.7b, $n = 3$, $l = 11$, $M_1 = 9$, and $M_2 = 6$, so

$d_1 \approx 0.27$ and $d_2 \approx 0.36$. Likewise, we can extract (d_1, d_2) from other existing circuits with known functionalities, so that the QUEKO benchmarks imitate some real-world circuits and, at the same time, have known optimal depths.

The construction of QUEKO, as shown in Algorithm 1, starts with checking the validity of input data by calculating the number of single-qubit and two-qubit gates M_1 and M_2. If $M_1 + M_2 < T$, then there would be too few gates to generate a circuit with depth T; if $M_1 + 2M_2 > N \cdot T$, then there would be too many gates for the given depth and coupling graph. We define the matching bound u of a graph G to be the minimal size of maximal matchings of G. This means we can find at least u edges in G that pair-wisely share no vertices. If $M_2 > u \cdot T$, then there could be too many two-qubit gates for the given depth and coupling graph. In short, if $M_1 + M_2 < T$, $M_1 + 2M_2 > N \cdot T$, or $M_2 > u \cdot T$, we return an error to reject the input data. Otherwise, we proceed to three phases: backbone construction, sprinkling, and scrambling.

3.1.1 Backbone Construction Phase

This phase "grows" a sequence of T gates, each depending on the previous one, constituting a dependency chain of length T. This chain serves as the "backbone" of the circuit. For example, we start from the coupling graph as Figure 3.1a, and pick three executable gates $\tilde{g}_1$, $\tilde{g}_2$, and $\tilde{g}_3$ whose spacetime coordinates are $(1, (p_0, p_1))$, $(2, p_1)$, $(3, (p_1, p_2))$. They constitute a dependency chain of length $T = 3$, since all of them act on p_1. This is shown in Figure 3.1b, where gates at different layers are put on different "slices" from left to right. The "backbone" is colored green. Note that if the given graph is directed, then we make sure that the direction of the two-qubit gate we choose is consistent with the direction of the corresponding edge. Thus, our construction also works for directed coupling graphs.

To be more rigorous,[1] we first choose a random node or edge of G as x_1. In every

[1]It may be worthwhile to review Section 1.3.1 to get familiar with some notation we use below.

Algorithm 1 QUEKO construction, part 1

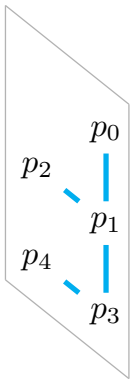
Input: a coupling graph $G = (P, E)$ with $|P| = N$ and its matching bound u, a depth target T, and a gate density vector (d_1, d_2).

Output: QUEKO benchmark $g_1 g_2 ... g_{M_1+M_2}$, where M_1 and M_2 are the numbers of single-gates and two-qubit gates, respectively.

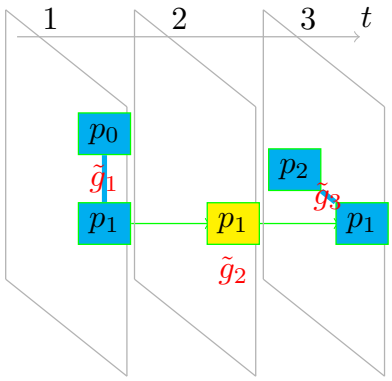
$M_1 \leftarrow \lceil d_1 \cdot N \cdot T \rceil$, $M_2 \leftarrow \lceil d_2 \cdot N \cdot T/2 \rceil$
if $M_1 + M_2 < T$ or $M_1 + 2M_2 > N \cdot T$ or $M_2 > u \cdot T$ **then**
    return error: input data not admissible
end if
$m_1 \leftarrow 0$, $m_2 \leftarrow 0$ ▷ how many single-qubit gates and two-qubit gates we have used
for $i = 1$ to T **do** ▷ Backbone construction phase
    $j \leftarrow \text{rand}(\{1, 2\})$ ▷ randomly decide single-qubit or two-qubit gate
    if $j = 2$ and $m_2 < M_2$ **then**
        $x_i \leftarrow \text{rand}(E)$
        while $i > 1$ and $x_i \cap x_{i-1} = \emptyset$ **do**
            $x_i \leftarrow \text{rand}(E)$
        end while
        $t_i \leftarrow i$, $m_2 \leftarrow m_2 + 1$
    else
        $x_i \leftarrow \text{rand}(P)$
        while $i > 1$ and $x_i \cap x_{i-1} = \emptyset$ **do**
            $x_i \leftarrow \text{rand}(P)$
        end while
        $t_i \leftarrow i$, $m_1 \leftarrow m_1 + 1$
    end if
end for

Algorithm 1 QUEKO construction, part 2

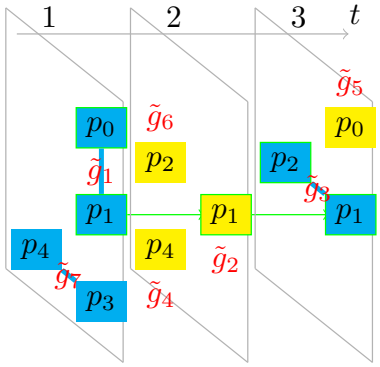
for $i = T + 1$ to $M_1 + M_2$ **do** ▷ Sprinkling phase
    $j \leftarrow \text{rand}(\{1, 2\})$
    if $j = 2$ and $m_2 < M_2$ **then**
        $(t_i, x_i) \leftarrow \text{rand}(\{1, 2, ..., T\} \times E)$
        while $\exists l \in \{1, ..., i\}$ such that $t_i = t_l$ and $x_i \cap x_l \neq \emptyset$ **do**
            $(t_i, x_i) \leftarrow \text{rand}(\{1, 2, ..., T\} \times E)$
        end while
        $m_2 \leftarrow m_2 + 1$
    else
        $(t_i, x_i) \leftarrow \text{rand}(\{1, 2, ..., T\} \times P)$
        while $\exists l \in \{1, ..., i\}$ such that $t_i = t_l$ and $x_i \cap x_l \neq \emptyset$ **do**
            $(t_i, x_i) \leftarrow \text{rand}(\{1, 2, ..., T\} \times P)$
        end while
        $m_1 \leftarrow m_1 + 1$
    end if
end for
$\tau \leftarrow$ a random map from P to Q ▷ Scrambling phase
for $i = 1$ to $M_1 + M_2$ **do**
    if $x_i = (p, p') \in E$ **then**
        $g_i \leftarrow \text{two-qubit gate}(\tau(x_i.p), \tau(x_i.p'))$
    else
        $g_i \leftarrow \text{single-qubit gate}(\tau(x_i))$
    end if
end for
sort g_i according to t_i, $i = 1$ to $M_1 + M_2$ ▷ Output phase
return $g_1 g_2 ... g_{M_1+M_2}$



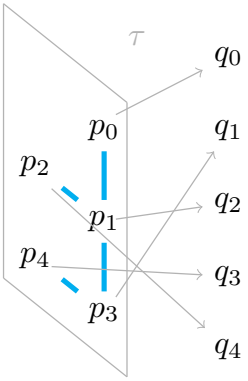
a) Coupling graph.



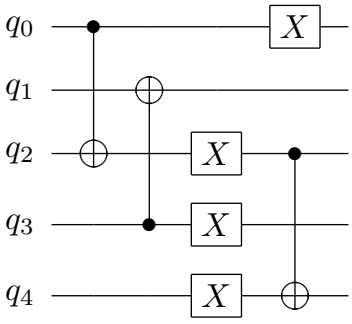
b) Backbone construction phase.



c) Sprinkling phase.



d) Scrambling phase.



e) Output circuit 1D diagram.

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[5];
cnot q[0],q[2]; //g1(g̃1)
cnot q[3],q[1]; //g2(g̃7)
x q[2]; //g3(g̃2)
x q[3]; //g4(g̃4)
x q[4]; //g5(g̃6)
cnot q[2],q[4]; //g6(g̃3)
x q[0]; //g7(g̃5)
```

f) Output QASM code (Inside the parentheses are the corresponding gates before the scrambling phase).

Figure 3.1: QUEKO construction.

iteration afterwards, we randomly choose x_k that overlaps with x_{k-1}. Thus, $x_k \cap x_{k-1} \neq \emptyset$, which enforces $t_k > t_{k-1} = k-1$ by dependency constraint. On the other hand, since $\tilde{g}_k$ is executable, it can at most take a single layer, i.e., the optimal $t_k = t_{k-1} + 1 = k$. Gate sequence $\tilde{g}_1, \tilde{g}_2, ..., \tilde{g}_T$ constitutes a dependency chain of length T. Because of this "backbone", the final depth of the scheduled circuit cannot be lower than T. Note that we do not need to use any SWAP gates for backbone construction.

3.1.2 Sprinkling Phase

The backbone construction phase uses T gates in total, we then randomly "sprinkle" the rest $M_1 + M_2 - T$ gates, e.g., $\tilde{g}_4, \tilde{g}_5, \tilde{g}_6, \tilde{g}_7$ shown in Figure 3.1c. We randomly select spacetime coordinates (t_i, x_i), $(1 \leq t_i \leq T)$ that do not overlap with any existing gates with time coordinate t_i. Since all the time coordinates of gates sprinkled are less or equal to T, the backbone is not lengthened in this phase. After sprinkling, a circuit with gates $\tilde{g}_1 ... \tilde{g}_{M_1+M_2}$ is created. Its gates are all executable; its depth is T; its gate density vector approximates (d_1, d_2). (There could be minor rounding errors in the ceiling function.)

It is worthy of noting that though only one longest dependency chain is explicitly generated in the backbone construction phase, the sprinkling phase may implicitly generate more. For example, $\tilde{g}_4$ depends on $\tilde{g}_7$; if we "sprinkles" a gate on p_4 at layer 3, then another dependency chain of length 3 would exist in the output circuit. The higher the gate densities, the more likely that these implicit longest dependency chains are generated.

3.1.3 Scrambling Phase

As shown in Figure 3.1d, we generate a random map τ from physical qubits to program qubits and apply τ to the space coordinates of $\tilde{g}_1 \tilde{g}_2 ... \tilde{g}_{M_1+M_2}$. For instance, $x_1 = (p_0, p_1)$, so the resulting gate g_1 is a two-qubit gate on program qubits $\tau(p_0) = q_0$ and $\tau(p_1) = q_2$; $x_7 = (p_3, p_4)$, so g_7 is a two-qubit gate on $\tau(p_3) = q_1$ and $\tau(p_4) = q_3$; g_6 is a single-qubit

gate on $\tau(p_2) = q_4$... The specific types of single-qubit gates and two-qubit gates are not important, since QUEKO is only for layout synthesis, not for circuit optimization. For simplicity, we use X as the single-qubit gate and CNOT as the two-qubit gate.

3.1.4 Output

Sort the gates $g_1 g_2 ... g_{M_1+M_2}$ according to the time coordinates to transfer the timing information originally in these time coordinates to the relative order inside the output gate list. The result is a QUEKO benchmark, as shown in Figure 3.1e and Figure 3.1f.

As we have proven, the depth of the output circuit is at least T because of the backbone. A layout synthesis tool can meet the optimum by finding the initial mapping that is the inverse of the scrambling map τ. Therefore, QUEKO circuits have known optimal depth T.

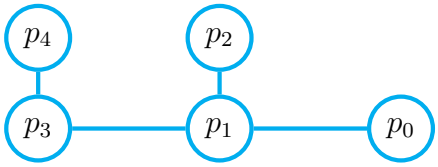
Note that QUEKO circuits also have known optimal gate count M_1+M_2. Since we assume that, in layout synthesis, all the input gates need to be executed, the result produced by the tools has at least as many gates as the QUEKO circuit. The optimal gate count is also met with the optimal initial mapping τ^{-1}, since no SWAP gates are needed in this case.

3.2 Optimality Study with QUEKO

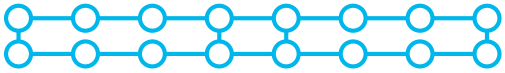
3.2.1 Experimental Setup

To evaluate layout synthesis tools with QUEKO, coupling graphs, depths and sizes, and gate density vectors are required. We specify the choice of these parameters and the choice of tools to evaluate in this subsection. Because of the randomness in our construction, we generate ten QUEKO benchmarks for each triplet of the parameters. These benchmarks along with the generating script are made open-source[2] under the BSD license. For evaluation, we fed each one of these benchmarks to four layout synthesis tools, as shown in Figure 3.3. All

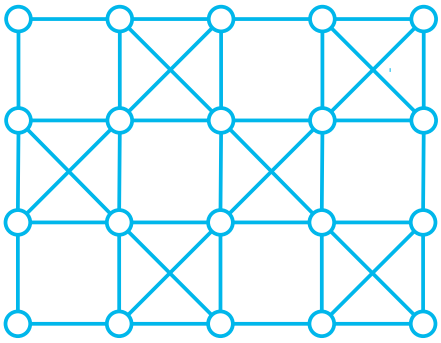
[2] https://github.com/UCLA-VAST/QUEKO-benchmark



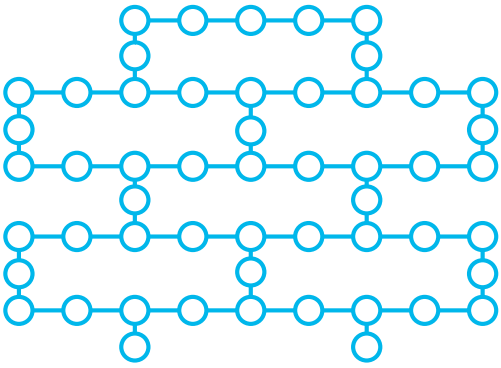
a) IBM's Ourense coupling graph.



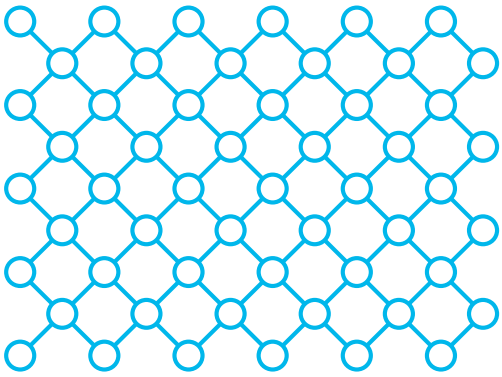
b) Rigetti's Aspen-4 coupling graph.



c) IBM's Tokyo coupling graph.



d) IBM's Rochester coupling graph.



e) Google's Sycamore coupling graph.

Figure 3.2: Examples of coupling graphs for quantum architectures.

the experiments were run on a Ubuntu 16.04 server, which has two Intel Xeon E5-2699v3 as CPUs and 128GB main memory.

3.2.1.1 Coupling Graph

We used representative coupling graphs from three quantum hardware providers. Sycamore from Google [AAB19], Tokyo and Rochester from IBM[3], and Aspen-4 from Rigetti[4]. The

[3] https://quantum-computing.ibm.com/

[4] https://www.rigetti.com/qpu

coupling graphs of these architectures are shown in Figure 3.2. Sycamore has 54 qubits, of which 53 are active; Rochester also has 53 qubits but with sparser connectivity. Aspen-4 has 16 qubits, and Tokyo has 20 qubits with greater connectivity. We have only listed superconducting architectures, but QUEKO directly generalizes to other technologies as long as the basic quantum gates are single-qubit and two-qubit gates, and the coupling graph is fixed.

3.2.1.2 Depth and Size

We constructed two sets of benchmarks with different depth ranges. The corresponding size of these benchmarks can be deduced from the depth and the gate density vector, as shown in Algorithm 1. The first set has depths from 5 to 45, which is the *near-term feasible benchmarks* (B_{NTF}). In fact, one of the largest quantum circuits executed at the time has depths 41 [AAB19], which is about the same with the upper bound of B_{NTF}. We intended to find out the layout synthesis performance within the current execution capacity. The sizes of B_{NTF} benchmarks range from 37 to 1727 quantum gates. The second set of benchmarks, denoted as B_{SS} has depth from 100 to 900 which are *benchmarks for scaling study*. B_{SS} represents the performance of these tools when the decoherence time of quantum hardware improves in the future. The sizes of B_{SS} benchmarks range from 1136 to 34506 quantum gates.

3.2.1.3 Gate Density Vector

We picked two special gate density vectors in the experiment: $(0.51, 0.4)$ based on the quantum circuits used in Google's quantum supremacy experiment [AAB19], denoted "QSE" below, and $(0.27, 0.36)$ based on the Toffoli circuit, denoted "TFL" below. It is beneficial to study QSE, since it was the only circuit at the time with which experimental quantum computing has shown a clear advantage. We chose the TFL because existing logic synthesis

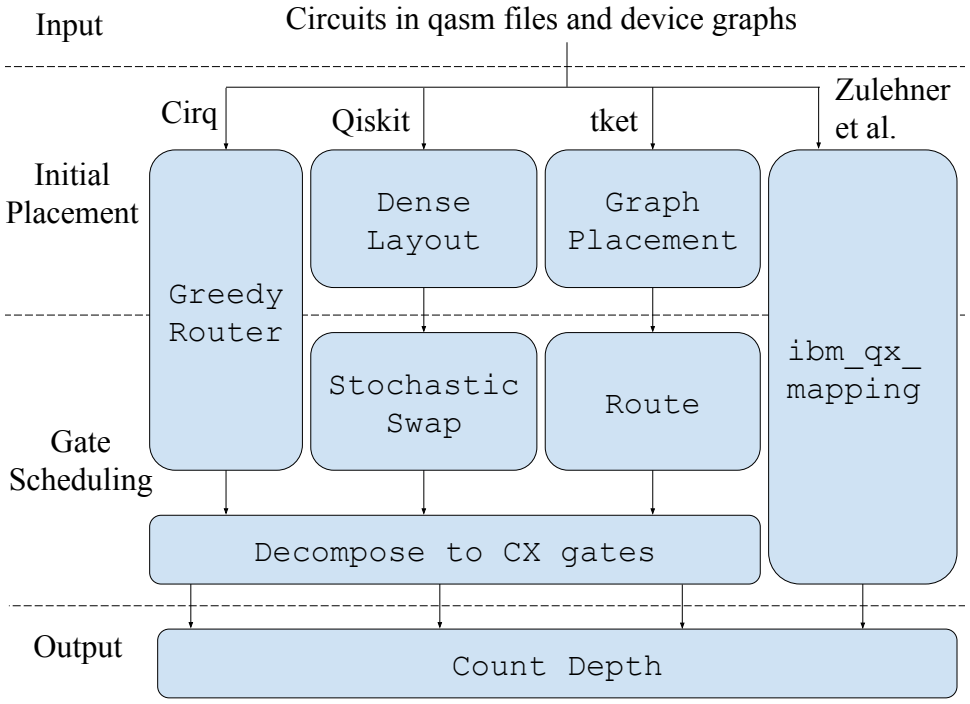


Figure 3.3: Workflow of the optimality experiments using QUEKO. 'CX gate' is also known as CNOT. Zulehner et al. is from [ZPW18].

algorithms are based largely on reversible logic synthesis, which uses TFL as a fundamental element [SPM03]. Some B_{NTF} have TFL density and others have QSE density. All B_{SS} have QSE density. We also swept through possible gate density vectors and generated *benchmarks for impact of gate density* (B_{IGD}).

3.2.1.4 Layout Synthesis Tools

At the time, Google, IBM, and Rigetti were considered front-runners of superconducting quantum computing. Inside their quantum programming frameworks (Cirq, Qiskit, and pyQuil), there are tools for layout synthesis. Unfortunately, pyQuil does not provide options to breakdown the whole compilation into optimization and layout synthesis, so pyQuil was excluded from the experiments. We also included a recent academic work from Zulehner et

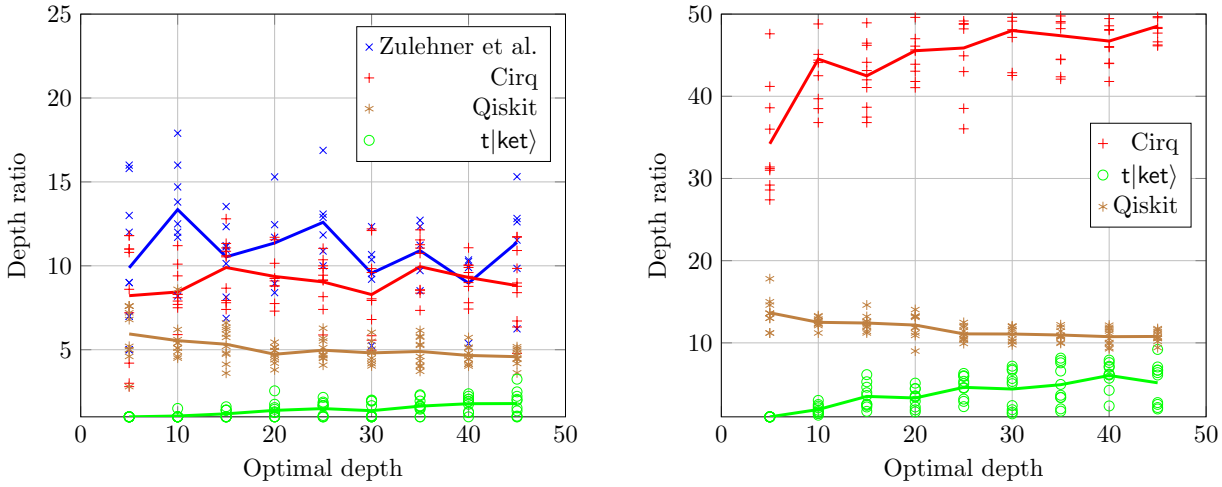
al. [ZPW18], which is open-source.

We used `greedy router` in Cirq version 0.6.0 as one of the layout synthesis tools, as shown in Figure 3.3. At the time, only one router named `greedy` had been released, which contains an initial placement policy, and a SWAP insertion policy based on heuristic search. Note that `greedy router` does not transform the gates into gates that are native on Google architectures so the resulting circuit it produces contains the original input gates and SWAP gates it inserts. For a fair comparison of depth, in all the experiments, we decompose SWAP gates inserted by the tools to three CNOT gates, like in Figure 1.8b.

Qiskit offers the most precise control over the so-called "transpiler". The transpilation is divided into individual passes, and users can define their own "pass manager" to make use of various transpiling modules that are offered. For the layout synthesis problem, there are `Layout` modules generating initial mapping and `Swap` modules inserting SWAP gates to the circuit to enable two-qubit gates. Among the various combinations, we chose `DenseLayout` and `StochasticSwap` as shown in Figure 3.3, which seemed to have the best overall performance at the time. `DenseLayout` maps the program qubits to an area on the coupling graph with dense connections. `StochasticSwap` perturbs the distance matrix of physical qubits and performs heuristic search for SWAP gates. The version of Qiskit in the experiments is 0.14.1.

Another highly competitive router, t|ket⟩, comes from Cambridge Quantum Computing, which becomes part of Quantinuum. `Graph Placement` uses graph monomorphism to derive initial mapping. `Route` performs heuristic search for SWAP gates. We used t|ket⟩ version 0.4.1 in the experiments.

Since all the tools evaluated use heuristics at some stages, sub-optimality is expected. Note that we only use default setup on all the modules in all the tools. Changing setup parameters in some of these modules specifically for QUEKO benchmarks may lead to better performance in the following experiments, but may lead to worse performance on other circuits.



a) Smaller architecture (Aspen-4), sparser circuits (TFL)

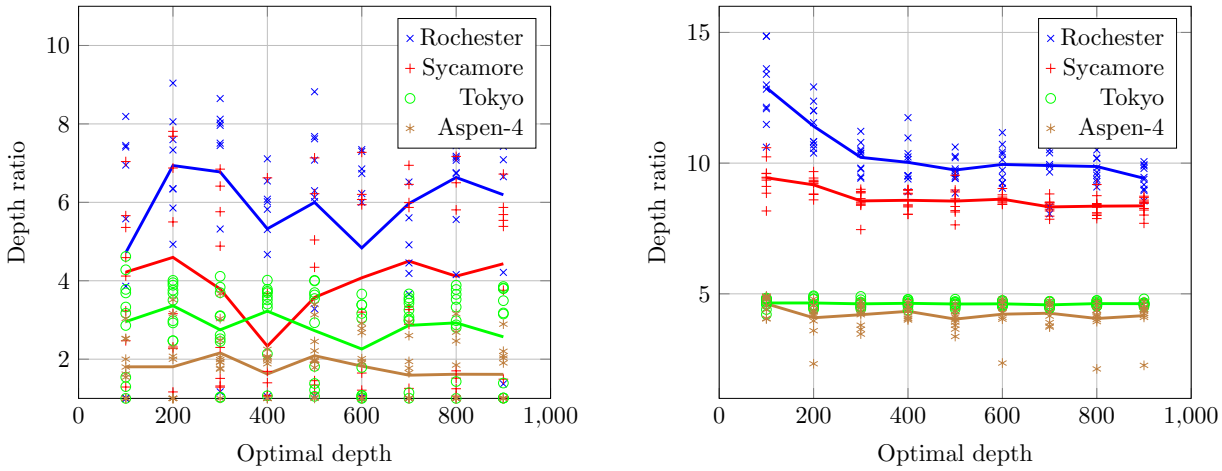
b) Larger architecture (Sycamore), denser circuits (QSE)

Figure 3.4: Performance of layout synthesis tools on B_{NTF} (lines are average).

3.2.2 Experimental Results

3.2.2.1 Performance on B_{NTF}

In Figure 3.4, the horizontal axis is the optimal depth, and the vertical axis is the depth ratio, which is the depth of layout synthesis result divided by the optimal depth T. In the case of a smaller architecture (Aspen-4) and sparser circuits (TFL), the optimality gap on average is about 12x for [ZPW18], 10x for Cirq, 5x for Qiskit and 1.5x for t|ket⟩. In the case of a larger architecture (Sycamore) and denser circuits (QSE), the optimality gap on average is about 11x to 14x for Qiskit. The optimality gaps of Cirq and t|ket⟩ grow with depth correspondingly from 35x to 50x and from 1x to 5x. Zulehner et al. is not in Figure 3.4b, because for the larger architecture, it took so much memory that the operating system constantly killed it before finishing. This also happens sometimes for the smaller architecture experiments, so there are fewer blue data points than the other types of points in Figure 3.4a.



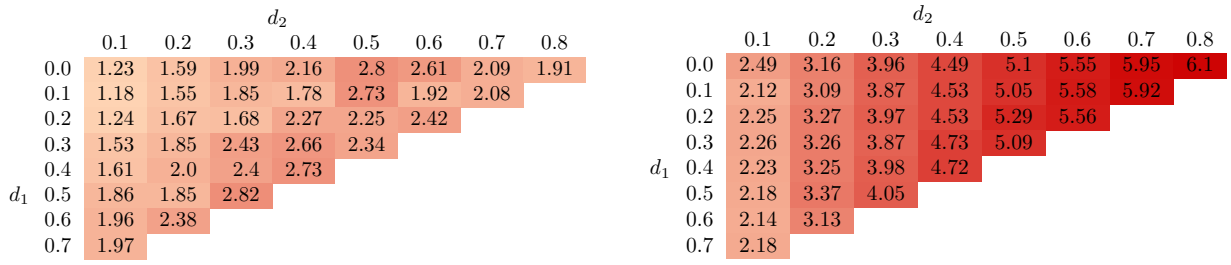
a) t|ket⟩ scaling behavior

b) Qiskit scaling behavior

Figure 3.5: t|ket⟩ and Qiskit performance on B_{SS} (lines are average).

3.2.2.2 Performance on B_{SS}

We studied further scaling of t|ket⟩ and Qiskit on different architectures as shown in Figure 3.5. The optimality gaps on average by t|ket⟩ are about 5x to 7x for Rochester, 3x to 4x for Sycamore, 3x for Tokyo, and 2x for Aspen-4. Note that for a fixed depth and a fixed architecture, the optimality gaps by t|ket⟩ varies rather widely. t|ket⟩ managed to find the optimal mappings for some QUEKO benchmarks. In general, as the depth increases, the depth ratio by Qiskit decreases at first and then converges to a value. The reason for this phenomenon may be that as the circuit deepens, the influence of initial placement gets smaller than the influence of SWAP insertion. For Qiskit, the optimality gaps on Rochester decreased from 13x to 10x on average; on Sycamore, Tokyo and Aspen-4 are about 8x, 5x, and 4x on average. It can be seen that larger architectures (Rochester and Sycamore) bring about larger optimality gaps. If the number of physical qubits is close, then richer connectivity (Sycamore versus Rochester) brings about smaller optimality gaps.



a) t|ket⟩ performance in depth ratio

b) Qiskit performance in depth ratio

Figure 3.6: t|ket⟩ and Qiskit performance on B_{IGD} (data are 10-time average).

3.2.2.3 Performance on B_{IGD}

To better understand the impact of gate density on layout synthesis performance, we fixed the architecture to Tokyo and the depth to 45 and swept through possible gate densities. The results are shown in Figure 3.6. Fixing a column, the single-qubit gate density increases as we go down, Qiskit seems to be rather insensitive to this change, which is sensible since the single-qubit gates do not induce difficulty in layout synthesis. However, t|ket⟩ is still sensitive to this change. Both tools are more sensitive to the change in the horizontal direction than in the vertical direction. Since the challenge to layout synthesis comes mainly, if not solely, from the two-qubit gates, this result is expected. The depth ratio of t|ket⟩ decreases when the two-qubit gate density is very high. This is because when the circuit is dense with two-qubit gates, graph monomorphism algorithms can extract more information from the first few layers to narrow down better initial mappings.

CHAPTER 4

OLSQ: Optimal Layout Synthesis for Static Quantum Architectures

In this chapter, we present our formulation of layout synthesis for static architectures, which results in a tool, OLSQ [TC20]. We shall motivate the work by analyzing the opportunities in formulation, and then dive in our approach. Additionally, we introduce two important relaxations to OLSQ: 1) structuring circuits as gate blocks with a fixed qubit mapping, and transitions between these blocks, and 2) removing the dependency constraints when all gates commute.

Let us first review some notations using the running example of this chapter: layout synthesis of a quantum adder circuit [AMM13] in Figure 4.1 onto the coupling graph in Figure 4.2a. In the circuit, g_0 is an X gate on program qubit q_0; g_3 is a CNOT gate, on q_2 and q_3. To make a distinction between single-qubit and two-qubit gates, we separate them into two lists G_1 and G_2. For $g_l \in G_1$, we use $g_l.q$ to denote the program qubit it operates on; for $g_l \in G_2$, we use $g_l.q$ and $g_l.q'$. For example, $g_0.q = q_0$, $g_5.q = q_1$, $g_3.q = q_2$, $g_3.q' = q_3$.

The quantum architecture is described by a coupling graph $G = (P, E)$, where $P = \{p_0, p_1, ..., p_{N-1}\}$ is the set of physical qubits and $E = \{e_0, e_1, ..., e_{K-1}\}$ is the set of (undirected) connections between them. (There are, in total, N physical qubits and K edges.) The coupling graphs used in this paper are displayed in Figure 4.2. We denote an edge as $(e_k.p, e_k.p')$, e.g., in Figure 4.2a, $e_1.p = p_0$ and $e_1.p' = p_2$. In addition, fidelity information can be provided as three functions: $f_0 : P \to [0, 1]$ for measurements, $f_1 : P \to [0, 1]$ for single-qubit gates, and $f_2 : E \to [0, 1]$ for two-qubit gates.

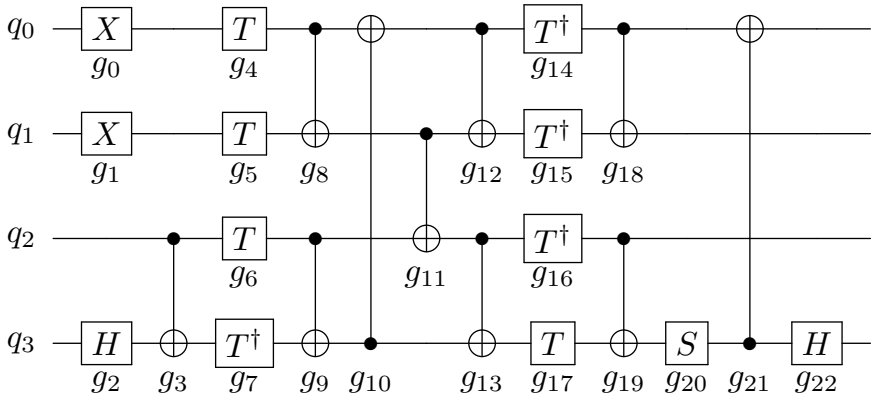
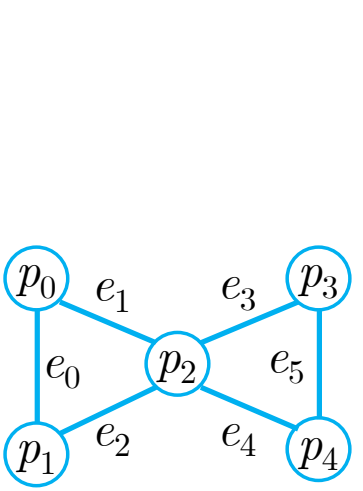
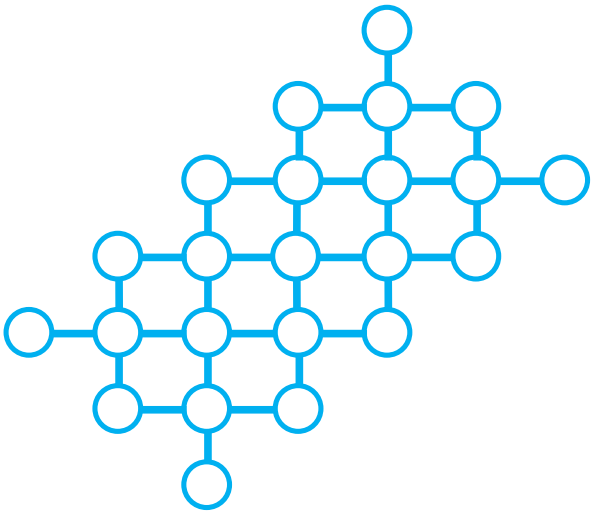


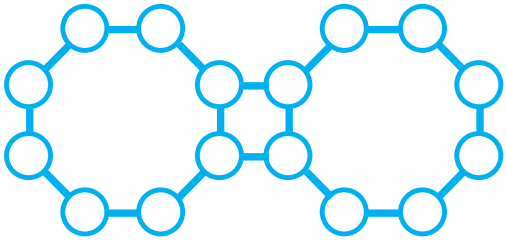
Figure 4.1: Circuit diagram for quantum adder.



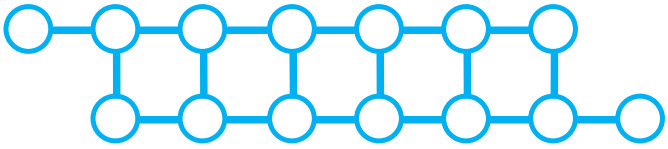
a) IBM QX2.



b) (Part of) Google Sycamore.



c) Rigetti Aspen-4.



d) IBM Melbourne.

Figure 4.2: Coupling graphs of some quantum architectures.

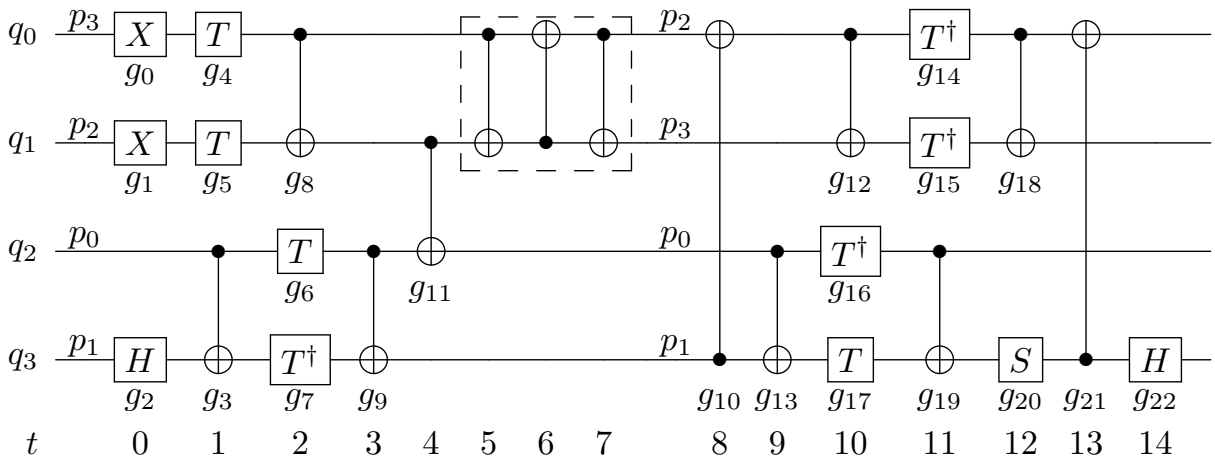


Figure 4.3: OLSQ layout synthesis result for the quantum adder circuit.

The output of layout synthesis is the *spacetime coordinates* (t_l, x_l) for all the input gates and the SWAPs inserted, and a final qubit mapping $\pi : Q \to P$ providing which physical qubit to measure for each program qubit. We show a layout synthesis result of the running example in Figure 4.3. In this diagram, gates are aligned according to their time coordinates, e.g., $t_0 = t_1 = t_2 = 0$ and $t_{10} = 8$; the space coordinates of single-qubit gates can be read off from the mapping displayed above the wires, e.g., q_0 is mapped to p_3 at time 0, so $x_0 = p_3$; the space coordinates of two-qubit gates can be deduced from the mapping, e.g., at time 8, q_3 and q_0 are mapped correspondingly to p_1 and p_2, so $x_{10} = e_2$ because e_2 connects p_1 and p_2. The mapping remains the same as the previous time step if it is not displayed, e.g., q_1 is still mapped to p_2 at time 1, so $x_5 = p_2$. Thus, the final mapping is just the last mapping displayed, $\pi(q_0) = p_2$, $\pi(q_1) = p_3$, $\pi(q_2) = p_0$, and $\pi(q_3) = p_1$. A SWAP consisting of three CNOTs is inserted on e_3 between p_2 and p_3. We use the last time step each SWAP takes as its time coordinate (in this case, 7).

The inserted SWAP in this example is absolutely necessary. The quantum program has two-qubit gates between q_0 and q_1 (g_8, g_{12}, and g_{18}), q_1 and q_2 (g_{11}), q_2 and q_3 (g_3, g_9, g_{13}, and g_{19}), and finally q_3 and q_0 (g_{10} and g_{21}). This means, without any SWAPs, the program qubits must be mapped to a set of physical qubits connected like a square. However, the

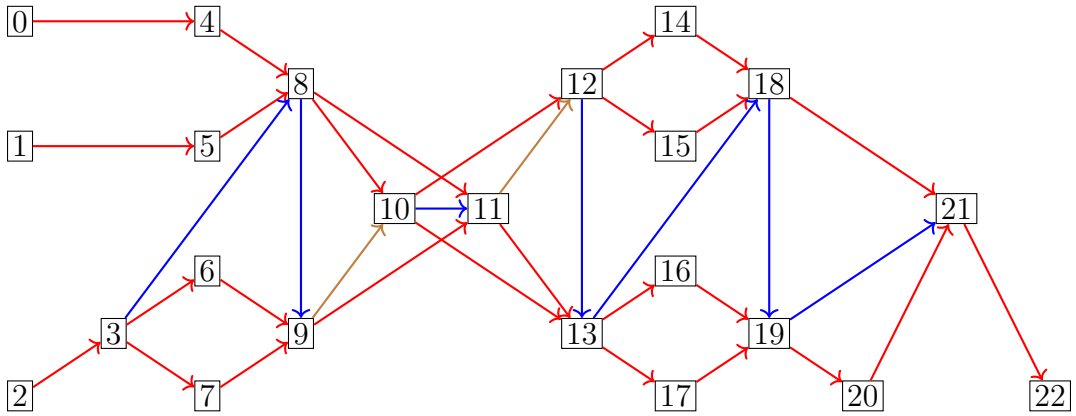


Figure 4.4: Immediate dependencies in the quantum adder circuit. Red arrows are used in OLSQ. Green arrows are those imposed implicitly by the approach of [WBZ19]. Brown means imposed by both OLSQ and [WBZ19].

coupling graph, Figure 4.2a, does not contain such a structure.

In layout synthesis, we need to avoid *collisions*: if two gates g_l and $g_{l'}$ act on the same qubit, then they cannot be executed at the same time, i.e., $t_l \neq t_{l'}$. For example, $t_5 \neq t_8$ because both g_5 and g_8 act on q_1. In general, we also have to respect *dependencies*: if g_l and $g_{l'}$ act on the same qubit and $l < l'$, then their relative order should not change, i.e., $t_l < t_{l'}$. If there are no gates between these two gates, it is called an *immediate dependency*. Immediate dependencies of the quantum adder circuit are shown as red and brown arrows in Figure 4.4.

4.1 Opportunities in Layout Synthesis Formulation

Prior to OLSQ, there have been multiple exact or optimal approaches to layout synthesis, as discussed in Section 1.3.3. We find that they generally process the quantum circuit either gate-by-gate or layer-by-layer, which imposes some unnecessary constraints than arranging the gates with dependency. We shall illustrate this point with the running example. In this case, [WBZ19] returns a solution with two SWAPs, but one SWAP would be enough as shown

in Figure 4.3. This is because when arranging gate-by-gate, there is an implicit dependency between each two-qubit gate and the one after it, which are shown as blue and brown arrows in Figure 4.4. Specifically, there is a dependency from g_{10} to g_{11}, which means that in their result $t_{10} < t_{11}$, but in the optimal solution, Figure 4.3, $t_{10} > t_{11}$. In fact, when we manually impose an additional dependency from g_{10} to g_{11} in our tool OLSQ (to be specified later in the chapter), it gives a solution with two SWAPs as well. In essence, the gate list is only one of the topological orderings of the dependency graph like Figure 4.4. If we use a gate-by-gate arrangement with the input gate list order, the result certainly respects all the dependencies, but some of these are unnecessary and may lead to sub-optimality. OLSQ actually reconstructs the dependency graph from the gate list and thus effectively explores *all* topological orderings. As for arranging layer-by-layer, in the layout synthesis instance above, suppose we assign g_8 and g_9 as the first layer, g_{10} and g_{11} as the second layer, g_{12} and g_{13} as the third layer. Then there must be at least one SWAP inserted between layer one and two; otherwise, there is a mapping that can satisfy all the two-qubit connections required by g_8, g_9, g_{10}, and g_{11}. This is to say the coupling graph contains a square-like structure, which is impossible in Figure 4.2a. Similarly, there must be at least one SWAP inserted between layer two and three. Thus, at least two SWAPs are required, which is sub-optimal compared to Figure 4.3.

4.2 Our Approach

In this section, we discuss preprocessing, the objectives, the variable encoding scheme, and the constraints of our tool OLSQ, optimal layout synthesizer for quantum computing. Then, we introduce some variations to the notion of time to make the synthesizer transition-based (TB-OLSQ), which greatly increases efficiency with little or no performance degradation. Lastly, we consider commutation to improve TB-OLSQ for QAOA circuits.

4.2.1 Preprocessing

From the input program, we derive a *collision list* C: if two gates g_l and $g_{l'}$ with $l < l'$, act on the same program qubit, then we append $(g_l, g_{l'})$ to C. By default, we use the collision list as the *dependency list* D. Users are also welcome to input their own D based on knowledge of the program. With the dependency list, we can also derive the longest dependency chain with $O(L^2)$ time. This serves as a lower bound of depth to the layout synthesis result because the dependency chain can only be lengthened by SWAPs and cannot be shortened in any case. We will need a time coordinate upper bound T in the formulation. In the hope of a depth-optimal result, we use the longest dependency chain length as T in the beginning.

We also need to extract some features of the coupling graph $G = (P, E)$ where P contains all the physical qubits and E contains all the edges between them. We compute an *overlapping edge pair set* O: $\forall e, e' \in E$, $e' \neq e$, if e and e' share some node, append the pair (e, e') to O. We also compute an incident edge set E_p for each node p: $\forall e \in E$, if $e = (\cdot, p)$ or $e = (p, \cdot)$, we append e to E_p. It is straightforward that $E_p \subset E$ and $\cup_{p \in P} E_p = E$.

4.2.2 Encoding Variables

- Mapping $\pi_{q,t}$: at time step t, program qubit q is mapped to physical qubit $\pi_{q,t} \in P$.
- Time coordinates t_l: gate g_l is being executed at time t_l, $0 \leq t_l \leq T - 1$.
- Space coordinates x_l: if $g_l \in G_1$, then program qubit $g_l.q$ is mapped to physical qubit $x_l \in P$; if $g_l \in G_2$, then the two physical qubits, to which $g_l.q$ and $g_l.q'$ are mapped, are connected by edge $x_l \in E$.
- Use of SWAP gate $\sigma_{k,t}$: if $\sigma_{k,t} = 1$, then there is a SWAP on edge e_k and the last time step it takes is t (as SWAPs may take multiple time steps); otherwise, $\sigma_{k,t} = 0$.

4.2.3 Constraints

Note that we differentiate variable assignment, =, and comparison, ==, in this chapter. The latter returns true if and only if the equality holds. There is an additional parameter S in our model which stands for the number of time steps a SWAP requires. S can be set according to different architectures. We set $S = 3$ as default, as seen in Figure 4.3.

4.2.3.1 Injective Mapping

Different program qubits should be mapped to different physical qubits at any specific time

$$\pi_{q,t} \neq \pi_{q',t} \quad \text{for } 0 \leq t \leq T-1,\ q, q' \in Q \text{ and } q \neq q' \tag{4.1}$$

4.2.3.2 Avoiding Collisions and Respecting Dependencies

$$t_l < t_{l'} \quad \text{for } (g_l, g_{l'}) \in D \tag{4.2}$$

4.2.3.3 Consistency between Mapping and Space Coordinates

There are two ways we can derive where a gate g_l is at physically: 1) directly through its space coordinate x_l; 2) indirectly from the mapping of the program qubit(s) it acts on at its time coordinate, i.e., $\pi_{g_l.q,t_l}$ for single-qubit gates; these two should be consistent.

$$(t_l == t) \Rightarrow (\pi_{g_l.q,t} == x_l) \quad \text{for } 0 \leq t \leq T-1, g_l \in G_1 \tag{4.3}$$

$$\begin{aligned}[(t_l == t) \wedge (x_l == e)] \Rightarrow \{&[(\pi_{g_l.q,t} == e.p) \wedge (\pi_{g_l.q',t} == e.p')] \vee \\ &[(\pi_{g_l.q,t} == e.p') \wedge (\pi_{g_l.q',t} == e.p)]\} \quad \text{for } 0 \leq t \leq T-1,\ g_l \in G_2,\ e \in E\end{aligned} \tag{4.4}$$

4.2.3.4 Proper SWAP Insertion

Since a SWAP takes S time steps, before time $S-1$, no SWAPs can finish:

$$\sigma_{k,t} = 0 \quad \text{for } 0 \le t \le S-2,\ 0 \le k \le K-1 \tag{4.5}$$

A SWAP cannot overlap with other SWAPs on the same edge:

$$(\sigma_{k,t} == 1) \Rightarrow (\sigma_{k,t'} == 0) \quad \text{for } S-1 \le t \le T-1,\ t-S+1 \le t' \le t-1,\ 0 \le k \le K-1 \tag{4.6}$$

If two edges overlap in space, the SWAPs on them cannot overlap in time:

$$(\sigma_{k,t} == 1) \Rightarrow (\sigma_{k',t'} == 0) \quad \text{for } S-1 \le t \le T-1,\ t-S+1 \le t' \le t,\ (e_k, e_{k'}) \in O \tag{4.7}$$

A SWAP should not overlap with any input single-qubit gates at any time:

$$\begin{aligned} &\{(t_l == t') \wedge [(x_l == e_k.p) \vee (x_l == e_k.p')]\} \Rightarrow (\sigma_{k,t} == 0) \\ &\quad \text{for } S-1 \le t \le T-1,\ t-S+1 \le t' \le t,\ 0 \le k \le K-1,\ g_l \in G_1 \end{aligned} \tag{4.8}$$

A SWAP on e_k should not overlap with any input two-qubit gates on the same edge or the edges that overlap at any time:

$$\begin{aligned} &[(t_l == t') \wedge (x_l == e_{k'})] \Rightarrow (\sigma_{k,t} == 0) \\ &\quad \text{for } g_l \in G_2,\ S-1 \le t \le T,\ t-S+1 \le t' \le t,\ (e_k, e_{k'}) \in O \text{ or } k' = k \end{aligned} \tag{4.9}$$

4.2.3.5 Mapping Transformations by SWAP Gates

Mapping at the next time step is the same with the current one if there are no SWAPs finished on all the edges in the incident edge set E_p:

$$\left[(\pi_{q,t} == p) \wedge \left(\bigwedge_{e_k \in E_p} \sigma_{k,t} == 0\right)\right] \Rightarrow (\pi_{q,t+1} == p) \quad \text{for } 0 \le t \le T-2,\ p \in P,\ q \in Q \tag{4.10}$$

If there is a SWAP finished at t, there can only be one. (Otherwise, the two SWAPs are on two edges that overlap. This case would be ruled out by Equation 4.7.) The mapping at

$t+1$ is then transformed by the SWAP:

$$\begin{gathered}
[(\pi_{q,t} == e_k.p) \wedge (\sigma_{k,t} == 1)] \Rightarrow (\pi_{q,t+1} == e_k.p') \\
[(\pi_{q,t} == e_k.p') \wedge (\sigma_{k,t} == 1)] \Rightarrow (\pi_{q,t+1} == e_k.p) \\
\text{for } 0 \leq t \leq T-2,\ 0 \leq k \leq K-1,\ q \in Q
\end{gathered} \tag{4.11}$$

4.2.4 Objectives

With the set of variables defined above, it is easy to construct the common objectives. In fact, as long as a quantity can be defined from the above variables, it can be the objective.

1) Depth: $d \coloneqq \max_{0,1,\ldots,L-1} t_l$. We do not need to consider the time coordinates of the inserted SWAPs because, if a SWAP has an even larger time coordinate than d defined above, it finishes after all the input gates, thus has no effects on the program and should be ignored.

2) SWAP cost: $c \coloneqq \sum_{k=0}^{K-1} \sum_{t=0}^{T-1} \sigma_{k,t}$.

3) (log-) Fidelity:

$$\begin{aligned}
f \coloneqq & \sum_{p \in P} \log f_0(p) \left[\sum_{q \in Q} (\pi_{q,T-1} == p) \right] + \sum_{p \in P} \log f_1(p) \left[\sum_{g_l \in G_1} (x_l == p) \right] \\
& + \sum_{e \in E} \log f_2(e) \left[\sum_{g_l \in G_2} (x_l == e) \right] + \sum_{k=0}^{K-1} \log f_{\text{SWAP}}(e_k) \left[\sum_{t=0}^{T-1} \sigma_{k,t} \right],
\end{aligned} \tag{4.12}$$

where f_0, f_1, and f_2 are given as input to the layout synthesis problem. $f_{\text{SWAP}}(e)$ is the fidelity of a SWAP on edge e, which should be computed from the provided single-qubit and two-qubit gate fidelity, depending on how SWAPs are implemented on the specific architecture. In our case, a SWAP consists of three CNOT gates, so $\log f_{\text{SWAP}} = 3 \log f_2$. To use addition rather than multiplication in the objective, we take the log of f_0, f_1, and f_2. To be compatible with other data and variables, which are all integers, we scale up every log fidelity value by 1000 and round it to the nearest integer.

Table 4.1: Complexity of OLSQ and related works. L_2 is the two-qubit gate count. N is the physical qubit count. T is the time coordinate upper bound. I is the number of layers of gates. L is the total gate count. B is the number of gate blocks.

	internal solver	**number of variables**	**number of constraints**
[WBZ19]	SMT	$O(L_2 \cdot N!)$	$O(L_2 \cdot M \cdot N!)$
[BSA19]	ILP	$O(T \cdot (N^4 + I))$	$O(T \cdot (N^4 + I^2 + N \cdot L))$
OLSQ	SMT	$O(T \cdot N + L)$	$O(T \cdot N \cdot L)$
TB-OLSQ	SMT	$O(B \cdot N + L)$	$O(B \cdot N \cdot L)$

4.2.5 Complexity Analysis

In our formulation, there are in total $MT + 2L + KT + 2N + 2K$ variables. For regular planar graphs, which most coupling graphs are, the number of edges is usually asymptotically linear to the number of nodes. For example, in a grid, each edge connects two nodes and each node spans out four edges, so $K/N \approx 2$. Therefore, the total number of variables in our formulation is $O(NT)$ where N is the physical qubit count and T is the time coordinate upper bound. The total search space is then exponential to N and T. This is expected from the NP-hardness of the problem (Section 1.3.2). However, as shown in Table 4.1, this formulation still has exponentially fewer variables compared to [WBZ19] because they have a variable for each permutation of qubits at each time. OLSQ also has polynomially fewer variables than [BSA19], because they also require variables encoding the mapping from *pairs of* program qubits to pairs of physical qubits at each time step. Also, it is straightforward from Table 4.1 that our formulation reduces the number of constraints significantly.

4.2.6 Remark on the Optimality of OLSQ

After we passed the variables, one of the objectives, and the constraints to Z3 SMT solver [dB08], it would either return a model containing all the variable values that optimizes the given objective, or return 'unsatisfiable'. As mentioned before, we initially set the time coordinate upper bound T to the largest length of dependency chain. However, it may be the case that on the given architecture, it is impossible to find a solution with this upper bound. Thus, if the model is unsatisfiable, we geometrically increase T each time by $(1+\epsilon)$x until it is satisfiable. We set $\epsilon = 0.3$ in our experiment. This means that OLSQ is optimal up to a certain time coordinate upper bound T. For depth optimization, the optimality is guaranteed. However, for SWAP cost and fidelity optimization, sometimes increasing T even more can lead to better results. However, this is a very rare case as we shall see in the evaluations, especially when the longest dependency chain in the input quantum program is already of considerable length.

4.2.7 TB-OLSQ: Rethinking Time Coordinates

In the example of quantum adder, the time upper bound is $T = 15$. However, the mapping to physical qubits changed only once at time step 8. Thus, for any specific program qubit q, the variables $\pi_{q,t}$ are the same from $t = 0$ to 7 and $t = 8$ to 14. This is a huge redundancy. In addition, the total search space for the solver is exponential to N and T. Although a cutting edge quantum processor at the time had only $N = 53$ [AAB19], T as determined by quantum programs can easily grow to be quite large.

These two observations motivate us to improve efficiency of OLSQ by rethinking time coordinates. Instead of keeping $\pi_{q,t}$ for all t, it turns out that we can keep only these variables between two *transitions* of the qubit mapping. Formally, a transition is a set of parallel SWAP gates. In the quantum adder example, there is only one transition, and the transition consists of only one SWAP on edge $e_3 = (p_2, p_3)$. SWAPs on overlapping edges

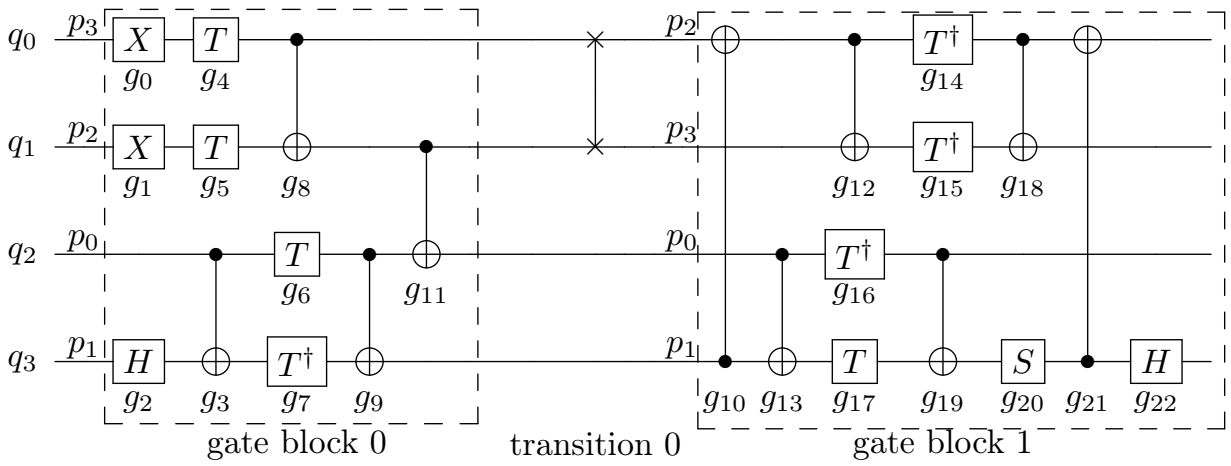


Figure 4.5: The quantum adder circuit in transition-based model. Two 'x's connected by a vertical line segment represent a SWAP gate.

cannot be in parallel, according to Equation 4.7, so e_1, e_2, e_4, and e_5 cannot be in the same transition with e_3. However, e_0 and e_3 together is a valid transition.

Now, we consider a new model of execution which separates the input gates and the inserted SWAPs: executing some input gates, then a few SWAPs to make transition(s) in mapping, execute some more input gates, and make other transition(s), ... We can have consecutive transitions without executing any input gates in between. This model is similar to the one in [WBZ19], but gates later in the input can appear at the front as long as permitted by dependency, which they do not allow. The quantum adder example in this model is shown in Figure 4.5, where we first execute the gates in gate block 0, then a SWAP to make transition 0, finally the gates in gate block 1. In this transition-based model, there is no notion of precise time. Instead, all the gates in a gate block are at the same coarse-grain time step and share the same mapping. This way, the number of mapping variables greatly decreases. In this particular example, there are only 8 mapping variables compared to the original 60 mapping variables.

With slight changes of formulation, OLSQ can be made transition-based. We change the $<$ in Equation 4.2 to $\leq$, since now even if two input gates have dependency, they can still be

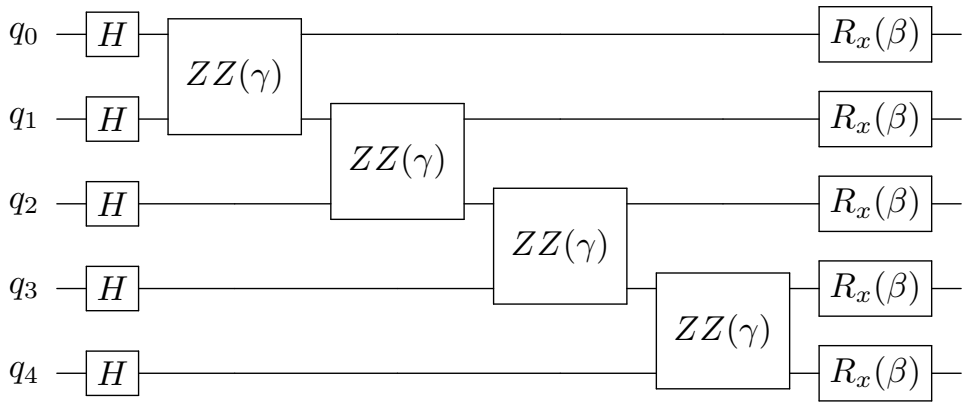
assigned to the same gate block, meaning their coarse-grain time coordinates can still be the same. We set $S = 1$ and remove Equation 4.8 and Equation 4.9. SWAP gates are separated from input gates in the current model, so we do not need to consider overlaps of input gates with them. $S = 1$ because we are using a coarse-grain time model.

The coarse-grain time upper bound T is initially set to 1, so the solver will search for a solution without any transition. If the solver returns 'unsatisfiable', we will increase T by 1 each time until it finds a solution that optimizes the given objective. The value of depth would just be $T - 1$, since there are exactly $T - 1$ gate blocks in the resulting circuit. If SWAP cost or fidelity is set as objective, TB-OLSQ will find the optimal solution that has up to $T - 1$ transitions. Just like OLSQ, there may be better solutions if T is increased even more. After a solution such as Figure 4.5 is returned, we can use as-soon-as-possible (ASAP) scheduling to derive all the exact time coordinates of the gates. After scheduling, the resulting format is exactly the same as that of OLSQ. We shall show that TB-OLSQ produced near-optimal solution with orders-of-magnitude speedup compared to OLSQ.

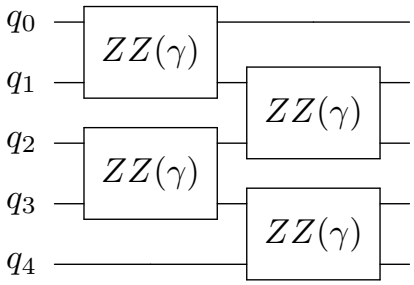
4.2.8 QAOA-OLSQ: Removing False Dependencies

One of the most important concepts in quantum mechanics is commutation. In the case of quantum computing, if two gates g_l and $g_{l'}$ $(l < l')$ commute, we can change their relative order without invalidating the whole program. This means that even if they consecutively act on the same qubit q_m, t_l is not necessarily smaller than $t_{l'}$. Since commutation is purely logical and has nothing to do with the architecture, one may think that it is solely the job of logic synthesis to experiment with the commutation relations. Like many other related works, both OLSQ and TB-OLSQ assume that any commutation is performed prior to layout synthesis and thus will indeed add $(g_l, g_{l'})$ as a dependency, eliminating the possibility of $t_l > t_{l'}$. However, it turns out that more knowledge of dependencies in layout synthesis is very beneficial — especially on the QAOA programs [FGG14, HWO19].

The quantum circuit in QAOA first sets the qubits to the equal superposition state by



a) Original QAOA program.



b) Better phase separation.

Figure 4.6: Sketch of the QAOA circuit.

applying H gates on all the qubits. Then it goes into many iterations. Each iteration has two stages: phase separation and mixing. A simple QAOA program with only one iteration is shown in Figure 4.6a [GJE20]. Phase separation is implemented by a few ZZ gates, which are two-qubit gates with a parameter γ; mixing is implemented by R_x gates on all the qubits. The specific functions of these gates are not of concern to layout synthesis; what matters is on which qubit(s) they act. Since single-qubit gates are always executable, the mixing stage does not require layout synthesis. However, phase separation may contain a ZZ gate on any pair of qubits, which means that layout synthesis is required to move the non-adjacent qubits together when a ZZ gate needs to act on them.

If we input the QAOA program, Figure 4.6a, and IBM QX2 coupling graph, Figure 4.2a,

to OLSQ or TB-OLSQ, the best result is probably just an identity mapping and the output looks the same with Figure 4.6a. OLSQ and TB-OLSQ cannot reduce depth because the four ZZ gates all depend on the gate before it, imposing the default dependencies. The particularity of QAOA is that all the ZZ gates commute, which means even both $ZZ(\gamma)(q_2, q_3)$ and $ZZ(\gamma)(q_3, q_4)$ act on q_3, the latter can actually commute 'through' the former. As a result, we have a better phase separation subroutine, as shown in Figure 4.6b, which has smaller depth. Current qubits still have short 'lifetimes' and many QAOA applications have large numbers of iterations, so reducing depth is crucial.

We can improve TB-OLSQ for the phase separation stage of QAOA with the knowledge of commutation. Previously, we treated every collision as a dependency. However, in the phase separation stage, all the ZZ gates are commutable, so none of the collisions are real dependencies. Thus, we simply remove the constraints in Equation 4.2 in TB-OLSQ. Up to this point, the result would be blocks of original ZZ gates with the fewest transitions possible to make all the qubit pairs adjacent required by these ZZ gates. Inside the ZZ gate blocks, there may be further opportunities to reduce depth with the help of commutation. Therefore, we input this result to OLSQ with depth as objective and, again, remove the constraints in Equation 4.2. Since the gates are already mapped to valid edges on the coupling graph, OLSQ does not need to insert any new SWAPs. Thus, we disable all the $\sigma_{k,t}$ variables in OLSQ for speedup. In the end, we derive the spacetime coordinates of the ZZ gates and the SWAPs. The depth of this result is optimized by the two passes of TB-OLSQ and OLSQ.

4.3 Evaluation

The evaluations were run on a Ubuntu 16.04 server with two Intel Xeon E5-2699v3 CPUs and 128GB memory. Wille et al. [WBZ19] and TriQ [MLM19] were built with Cmake 3.13.4 and GNU Make 4.1. The version of Python was 3.8.2. The versions of Python packages used were Cirq 0.8.0, Pytket 0.5.4, Qiskit 0.18.0, and Z3-Solver 4.8.7.0. We linked the Z3 library

contained in Z3-Solver package to Wille et al. and TriQ when building.

We selected a comprehensive set of benchmarks from various sources including [AMM13, NRS18, GJE20, TC21b]. We used the fidelity profile of IBM QX2, Figure 4.2a, and IBM Melbourne, Figure 4.2d, from [MLM19]. To evaluate fidelity, we input the result from different synthesizers to Qiskit, decomposed, and calculated the product of all gate fidelity. We made OLSQ, the benchmarks, and detailed results open-source.[1]

4.3.1 OLSQ versus Previous Optimal Approaches

The leading exact approach at the time was by Wille et al. [WBZ19]. The coupling graph in their original paper is directed, which means that CNOT gates can only execute in one direction. However, the edges in our formulation are bi-directional. For fair comparison, we input each bi-directional edge as two uni-directional edges to their software. We observe that, compared to the cases where directed coupling graphs are used, their software runs significantly faster. Thus, the runtime data we collect, for comparison with OLSQ, are smaller than those appeared in the original reference [WBZ19]. Since Wille et al. aims to minimize SWAP cost, it is most appropriate to compare it against OLSQ with SWAP cost as objective, denoted as OLSQ-SWAP below. We find that, on all instances of benchmark program and architecture, OLSQ-SWAP matches their performance and sometimes is even better. All the data for these evaluations are presented in Table 4.2.

Wille et al. have a set of variables denoting whether each qubit permutation is performed between two two-qubit gates. So, the number of variables in their formulation is proportional to $N!$, where N is the physical qubit count. This complexity explosion can be observed from the two examples in Figure 4.7, where the runtime of [WBZ19] is nearly exponential. The runtime of OLSQ-SWAP does not show this exponential growth. Wille et al. also rely on pre-processing to derive a function from each qubit permutation to its SWAP cost. Suppose each

[1] https://github.com/UCLA-VAST/OLSQ

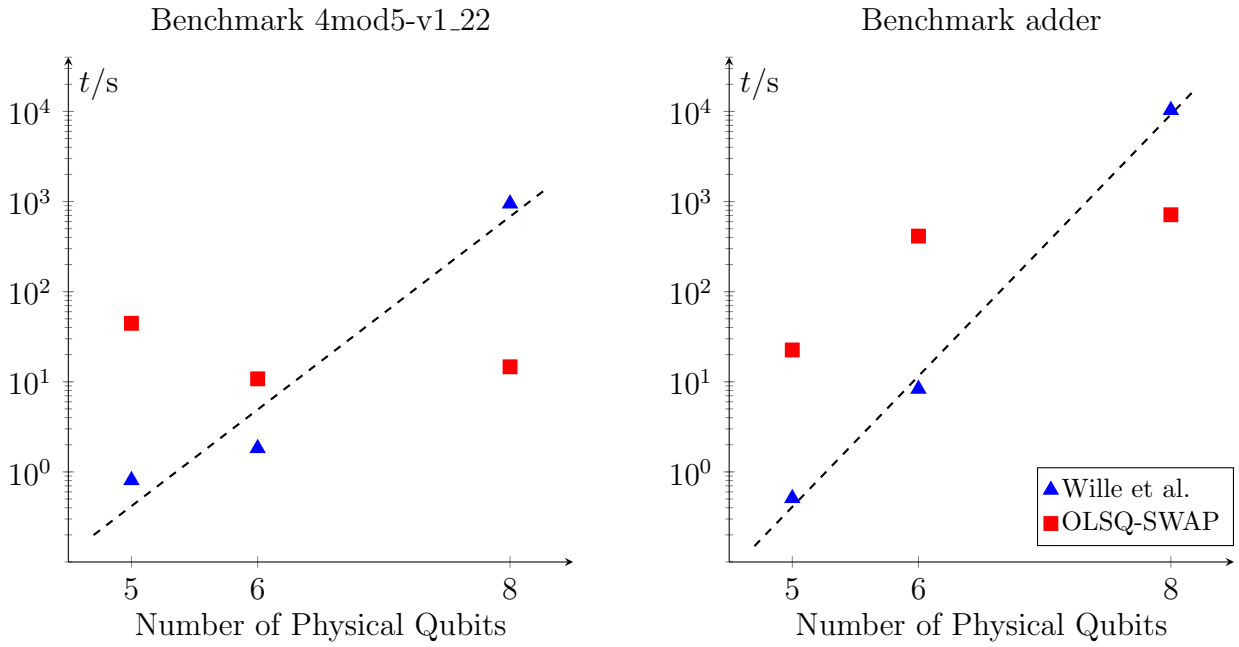


Figure 4.7: Runtime scaling of Wille et al. [WBZ19] and OLSQ-SWAP. The devices with 5, 6, 8, and 16 qubits are IBM QX2, a 2 by 3 grid, a 2 by 4 grid, and Rigetti Aspen-4. Dashed line is an exponential fit of Wille et al.'s results.

function value takes 4 byte of memory, then, even for device with qubit count $N = 12$, the memory required is 2 terabytes. As examples, we evaluate three layout synthesis instances of QUEKO benchmarks for Aspen-4, Figure 4.2c. All these evaluations of [WBZ19] are aborted because our server runs out of the memory limit we set (32GB). In contrast, OLSQ-SWAP reaches the optimal SWAP cost (0) within a relatively short period of time.

Table 4.2: Evaluation of OLSQ. M is program qubit count of the program; c is additional CNOT count; d is depth; f is fidelity (up to three digit precision); t is compilation time (up to one significant digit precision, 'E' means 10 exponential). We do not have a fidelity profile of the architectures with *, so we use a uniform fidelity profile based on the average fidelity of IBM QX2.

			Wille et al. [WBZ19]				OLSQ-SWAP				OLSQ-Depth				OLSQ-Fidelity			
Program	M	**Architecture**	**c**	**d**	**f**	**t**	**c**	**d**	**f**	**t**	**c**	**d**	**f**	**t**	**c**	**d**	**f**	**t**
or	3	IBM QX2	0	9	0.594	0.2	0	9	0.589	5	0	9	0.625	4	0	9	0.625	5
adder	4	IBM QX2	6	18	0.335	1	3	16	0.407	40	3	16	0.391	40	3	16	0.431	2E3
adder	4	Grid2by3*	0	12	0.462	2	0	12	0.462	10	0	12	0.462	10	0	12	0.462	10
adder	4	Grid2by4*	0	12	0.462	1E3	0	12	0.462	10	0	12	0.462	10	0	12	0.462	20
qaoa5	5	IBM QX2	0	15	0.470	0.3	0	15	0.466	10	3	15	0.389	10	0	15	0.476	10
mod5mils_65	5	IBM QX2	6	28	0.290	2	6	28	0.299	1E2	12	25	0.198	90	6	28	0.301	7E2
4gt_13_92	5	IBM QX2	0	39	0.160	2	0	39	0.171	2E2	12	39	0.107	1E2	0	39	0.171	3E2
4mod5-v1_22	5	IBM QX2	3	16	0.313	0.5	3	16	0.391	20	6	16	0.352	20	3	16	0.391	30
4mod5-v1_22	5	Grid2by3*	9	22	0.271	8	9	21	0.271	4E2	12	21	0.236	1E2	9	22	0.271	3E3
4mod5-v1_22	5	Grid2by4*	9	18	0.271	1E4	9	22	0.271	7E2	9	21	0.206	2E2	9	24	0.271	8E2
queko_05_0	16	Aspen-4*	out of memory (32GB)				0	6	0.148	70	0	6	0.148	70	0	6	0.148	9E2
queko_10_3	16	Aspen-4*	out of memory (32GB)				0	11	0.076	8E2	0	11	0.076	8E2	0	11	0.076	4E3
queko_15_1	16	Aspen-4*	out of memory (32GB)				0	16	0.037	5E3	0	16	0.037	5E3	0	16	0.037	1E4

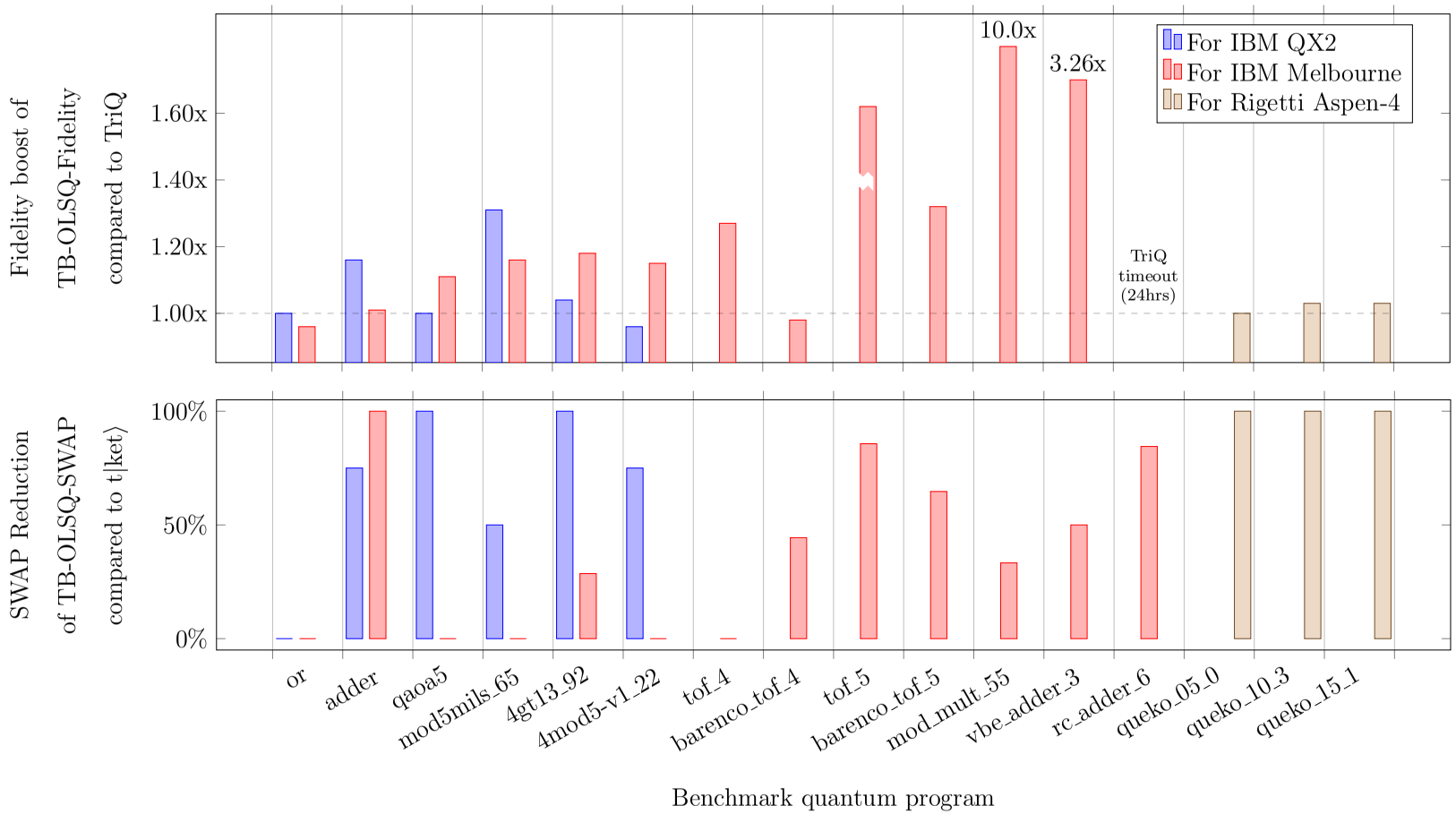


Figure 4.8: Evaluation of TB-OLSQ, t|ket⟩ [SDC20], and TriQ [MLM19].

Table 4.3: Evaluation of TB-OLSQ. M is program qubit count of the program; c is additional CNOT count; d is depth; f is fidelity (up to three significant digit precision). CNOT cost reduction is the difference of c of TB-OLSQ-SWAP and t|ket⟩ normalized by the c of t|ket⟩. Fidelity boost is the ratio of the f of TB-OLSQ-Fidelity and TriQ. We do not have a fidelity profile of the architectures with *, so we use a uniform fidelity profile based on the average fidelity of IBM QX2.

			t|ket⟩ [SDC20]			TB-OLSQ-SWAP			**CNOT**	TriQ [MLM19]			TB-OLSQ-Fidel.			**Fidelity**
Program	M	**Architecture**	**c**	**d**	**f**	**c**	**d**	**f**	**Reduc.**	**c**	**d**	**f**	**c**	**d**	**f**	**Boost**
or	3	IBM QX2	0	9	0.625	0	9	0.625	0	0	9	0.626	0	9	0.625	1.00x
adder	4	IBM QX2	12	27	0.246	3	16	0.439	75%	6	24	0.371	3	16	0.431	1.16x
qaoa5	5	IBM QX2	3	17	0.396	0	15	0.467	100%	0	16	0.475	0	15	0.476	1.00x
mod5mils_65	5	IBM QX2	12	34	0.203	6	29	0.249	50%	12	50	0.230	6	27	0.302	1.31x
4gt13_92	5	IBM QX2	21	64	5.72E-2	0	39	0.155	100%	0	48	0.165	0	39	0.171	1.04x
4mod5-v1_22	5	IBM QX2	12	29	0.282	3	16	0.384	75%	3	24	0.406	3	16	0.391	0.96x
queko_05_0	16	Aspen-4*	3	10	0.129	0	6	0.148	100%	0	6	0.148	0	6	0.148	1.00x
queko_10_3	16	Aspen-4*	45	45	9.59E-3	0	11	0.076	100%	0	11	0.076	0	15	0.074	1.03x
queko_15_1	16	Aspen-4*	114	58	1.96E-4	0	16	3.74E-2	100%	0	16	3.74E-2	0	19	3.62E-2	1.03x

(This table continues on the next page.)

(previous page)

Program	M	Architecture	t|ket⟩ [SDC20] c	d	f	TB-OLSQ-SWAP c	d	f	CNOT Reduc.	TriQ [MLM19] c	d	f	TB-OLSQ-Fidel. c	d	f	Fidelity Boost
or	3	Melbourne	6	18	0.350	6	16	0.276	0	6	23	0.364	6	17	0.350	0.96x
adder	4	Melbourne	3	14	0.216	0	12	0.363	100%	0	14	0.365	0	12	0.369	1.01x
qaoa5	5	Melbourne	0	15	0.340	0	15	0.114	0	0	17	0.381	0	15	0.424	1.11x
mod5mils_65	5	Melbourne	18	45	0.118	18	34	3.32E-2	0	21	65	8.88E-2	18	43	0.103	1.16x
4gt13_92	5	Melbourne	42	84	2.11E-3	30	68	5.36E-6	28.6%	39	116	1.48E-2	36	72	1.74E-2	1.18x
4mod5-v1_22	5	Melbourne	9	24	1.83E-4	9	16	0.131	0	9	38	0.215	9	25	0.247	1.15x
tof_4	7	Melbourne	3	53	2.04E-3	3	47	2.14E-3	0	6	50	9.57E-2	3	47	1.22E-1	1.27x
barenco_tof_4	7	Melbourne	27	91	3.93E-4	15	75	5.77E-4	44.4%	21	100	2.22E-2	24	78	2.18E-2	0.98x
tof_5	9	Melbourne	21	68	3.09E-4	3	62	1.00E-3	85.7%	6	63	2.85E-2	3	62	4.63E-2	1.62x
barenco_tof_5	9	Melbourne	51	146	3.02E-10	18	81	6.37E-6	64.7%	39	147	1.39E-3	39	93	1.83E-3	1.32x
mod_mult_55	9	Melbourne	36	78	1.60E-6	24	71	3.18E-5	33.3%	50	126	7.76E-4	24	68	7.77E-3	10.0x
vbe_adder_3	10	Melbourne	48	96	2.80E-8	24	59	1.00E-8	50.0%	45	124	1.53E-3	27	61	4.99E-3	3.26x
rc_adder_6	14	Melbourne	174	186	5.44E-22	27	80	1.89E-7	84.5%	timeout (24 hrs)			27	85	2.01E-6	N/A
Geometric Mean									69.2%							1.30x

4.3.2 TB-OLSQ versus Heuristic Approaches

One of the leading industry works at the time, t|ket⟩ [SDC20], mainly focuses on optimizing SWAP cost. We compare it with TB-OLSQ with SWAP cost set as objective, denoted as TB-OLSQ-SWAP below. One of the leading academic works at the time, TriQ [MLM19], mainly focuses on optimizing fidelity, so we compare it to TB-OLSQ with fidelity set as objective, denoted as TB-OLSQ-Fidelity below. The data are presented in Table 4.3 which is summarized by Figure 4.8.

For the relatively small-scale experiments with IBM QX2 and Aspen-4 architectures, we can compare OLSQ and TB-OLSQ by contrasting the corresponding rows in Table 4.2 and Table 4.3. We find that TB-OLSQ-SWAP has no observable degradation on CNOT cost compared to OLSQ-SWAP on these benchmarks, neither does TB-OLSQ-Fidelity compared to OLSQ-Fidelity on fidelity. This means that our transition-based approach is almost exact; it also hugely increases efficiency, e.g. solving queko_15_1 takes the former 9E4 seconds while taking the latter 30 seconds.

As displayed in Figure 4.8, compared to t|ket⟩, TB-OLSQ-SWAP often reduces the CNOT cost by large margins, 69.2% in geometric mean. Compared to TriQ, TB-OLSQ-Fidelity often increases the fidelity, even up to 10.0x on some larger programs. In some cases, TriQ has slightly better fidelity results by leveraging larger depths. This is expected since TB-OLSQ-Fidelity optimizes fidelity with the fewest transitions possible, but the number of transitions has no direct link to fidelity, so it is possible to find solutions with higher fidelity and more transitions. However, compared to TriQ, the fidelity loss of OLSQ in these cases remains less than 5%.

4.3.3 QAOA-OLSQ

Arute et al. [HSN21] is considered a leading work on QAOA implementation at the time, where t|ket⟩ is used to solve layout synthesis of QAOA programs for 3-regular graphs. We

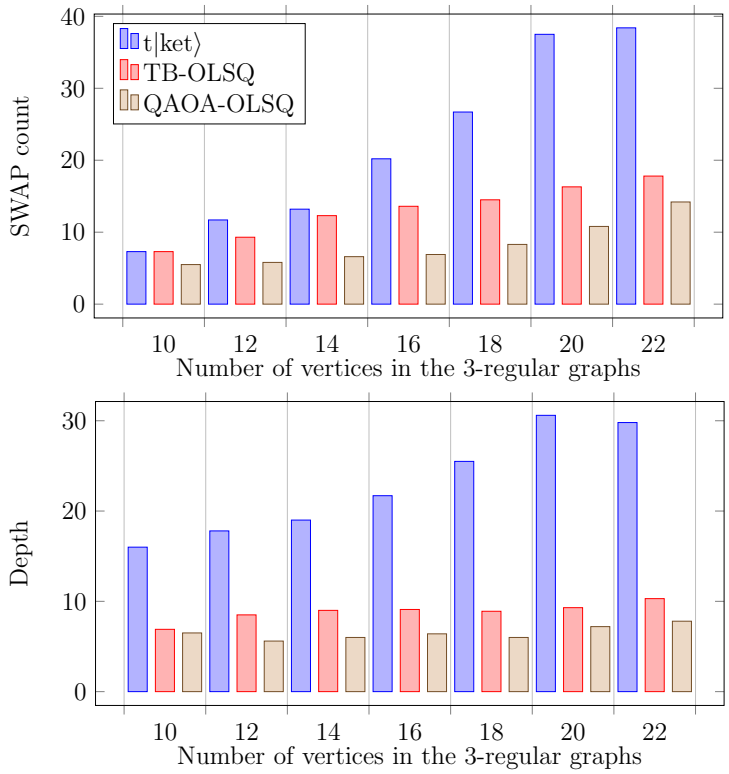


Figure 4.9: Evaluation of QAOA-OLSQ and t|ket⟩ [SDC20].

conduct evaluation with the same settings in [HSN21]: the coupling graph is part of Google Sycamore with 23 physical qubits, as shown in Figure 4.2b. We generate random 3-regular graphs with node count M from 10 to 22. (Graph theory shows that the product of M and the vertex degree must be even, so M must be even.) Then, for each edge (i, j) in a 3-regular graph, we append a corresponding gate $ZZ(q_i, q_j)$ to the phase separation stage. The phase separation is given to t|ket⟩, TB-OLSQ, and QAOA-OLSQ for layout synthesis and their results are fed to Cirq for statistics on SWAP cost and depth. We consider all ZZ gates and SWAP gates to have unity depth. As shown in Figure 4.9, even without considering the commutation relations, TB-OLSQ reduces depth by 59.5% in geometric mean and reduces SWAP cost by 29.4% in geometric mean. QAOA-OLSQ further reduces both depth and SWAP cost, by 70.2% and 53.8% in geometric mean compared to t|ket⟩. All these experimental data are provided in Table 4.4.

Table 4.4: Evaluation of QAOA-OLSQ. M is the node count of the 3-regular graphs and also the program qubit count. All the ZZ-phase gates and SWAP gates are seen to have unity depth. Depth and SWAP reduction is the difference between QAOA-OLSQ and t|ket⟩ normalized by t|ket⟩ result. All data are geometrical means of 10 random graphs of the same size and have one digit precision.

	t|ket⟩ [SDC20]		TB-OLSQ		**Depth**	**SWAP**	QAOA-OLSQ		**Depth**	**SWAP**
M	**Depth**	**SWAP**	**Depth**	**SWAP**	**Reduction**	**Reduction**	**Depth**	**SWAP**	**Reduction**	**Reduction**
10	16	7.3	6.9	7.3	56.7%	0	6.5	5.5	59.3%	23.6%
12	17.8	11.7	8.5	9.3	52.3%	20.4%	5.6	5.8	67.3%	46.2%
14	19.0	13.2	9.0	12.3	52.6%	6.8%	6.0	6.6	68.3%	48.0%
16	21.7	20.2	9.1	13.6	58.2%	32.7%	6.4	6.9	70.2%	62.6%
18	25.5	26.7	8.9	14.5	64.9%	45.7%	6.0	8.3	75.5%	65.7%
20	30.6	37.5	9.3	16.3	68.9%	57.7%	7.2	10.8	75.7%	68.8%
22	29.8	38.4	10.3	17.8	65.4%	53.6%	7.8	14.2	73.7%	61.8%
Geometric Mean					59.5%	29.4%			70.2%	53.8%

CHAPTER 5

OLSQ-GA: Optimal Layout Synthesis with SWAP Gate Absorption for Static Quantum Architectures

With OLSQ, we may insert some additional SWAPs to the circuit in layout synthesis. Would it be nice if these SWAPs were "free"? In some cases, this is possible by synthesizing other gates differently so that the functionality of the SWAP is integrated with existing gates. In Section 5.1, we explain how this "free lunch" SWAP works. In Section 5.2, we present the OLSQ-GA formulation which extends from OLSQ. In Section 5.3, we evaluate OLSQ-GA against previous works on QAOA benchmarks. In Section 5.4, we perform some analysis on the structure of optimal solutions and discuss our speedup strategies.

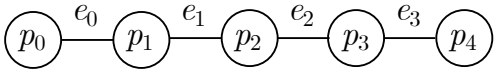
5.1 "Free Lunch" for Layout Synthesis: SWAP Absorption

Because of the generality of $U(4)$ gates, we can leverage SWAP absorption to reduce explicit SWAPs and depth. Suppose a gate W acts on two qubit p_i and p_j. Immediately before or after W, a SWAP on p_i and p_j is inserted. As illustrated in Figure 2.3, we can actually compute the matrix of SWAP $\cdot$ W and, after the layout synthesis, decompose the updated matrix. This way, the updated gate still has the decomposition in Figure 1.6, just with different single-qubit gates, which means the SWAP is absorbed into W with practically no cost. In some literature, this process is called 'implementing a mirrored gate' [JJB21].

Consider layout synthesis for Figure 5.1a, a program for general chemical simulation which consists of 10 two-qubit gates on 5 qubits, onto the coupling graph Figure 5.1b The

```
g0(q0, q1);  g1(q0, q2);  g2(q0, q3);  g3(q0, q4);  g4(q1, q2);
g5(q1, q3);  g6(q1, q4);  g7(q2, q3);  g8(q2, q4);  g9(q3, q4);
```

a) A general chemical simulation on 5 qubits. The quantum program is read from left to right, and from top to bottom.



b) The coupling graph of a linear architecture to run simulation.

Figure 5.1: Example layout synthesis problem.

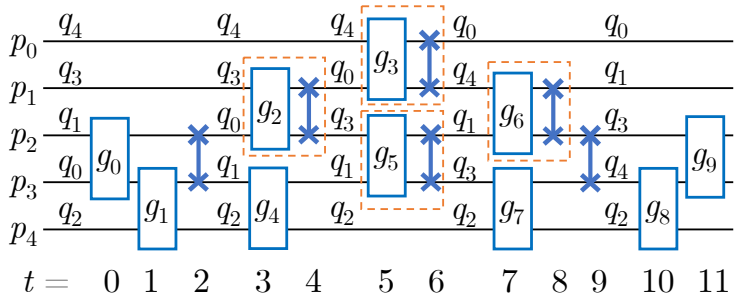
gates in this program are fermionic simulation gates with different parameters [FND20] that commute with each other.

$$\text{fSim}(\theta, \phi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -i\sin\theta & 0 \\ 0 & -i\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & e^{-i\phi} \end{pmatrix}. \tag{5.1}$$

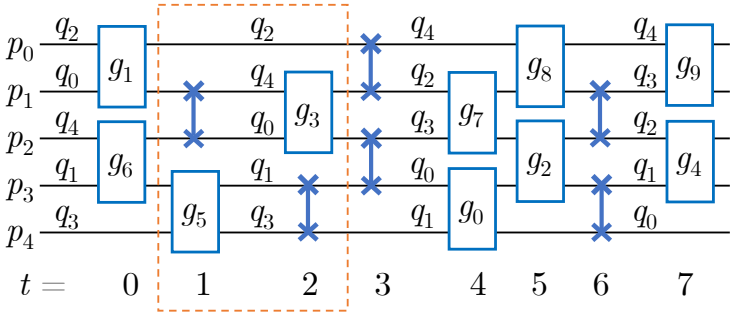
In Figure 5.2a, we show a solution by SABRE [LDX19] which does not exploit the commutations, i.e., all the dependencies are respected. However, in our simulation example, there are many commutations, indicating opportunities for depth and SWAP optimization.

The solution in Figure 5.2b produced by TB-OLSQ [TC20] is optimal with 6 SWAPs without consideration of absorption. There is an opportunity to reduce 2 SWAPs in the dashed box, with the absorption of the SWAP before g_3 and the SWAP after g_5. For g_5, we can compute the product

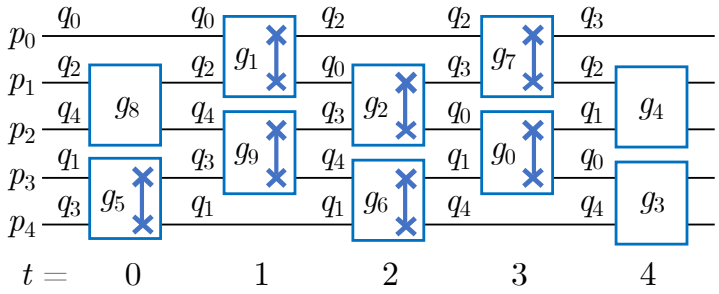
$$\text{fSwap} \cdot \text{fSim} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -i\sin\theta & \cos\theta & 0 \\ 0 & \cos\theta & -i\sin\theta & 0 \\ 0 & 0 & 0 & -e^{-i\phi} \end{pmatrix}, \tag{5.2}$$



a) SABRE [LDX19] solution with 6 SWAPs and depth 12. With post-processing, 4 SWAPs can be absorbed, and the depth becomes 9.



b) TB-OLSQ [TC20] solution with 6 SWAPs and depth 8. The two steps inside the dashed box can be combined with SWAP absorption as post-processing, then it would have 4 SWAPs and depth 7.



c) OLSQ-GA solution with no explicit SWAPs and depth 5. The SWAPs inside the boxes are absorbed.

Figure 5.2: Layout synthesis solutions of 5-qubit chemical simulation on a linear architecture. Connected crosses are SWAPs. At each time step, which program qubit locates at which physical qubit is shown.

and then pass this new matrix to the KAK decomposition subroutine. The notation 'fSwap' in the above equation is a fermionic SWAP used in chemical simulation. It is different from the normal form in Equation 1.7 in that the bottom right element is -1 instead of 1, but this does not affect the SWAP absorption technique. The fSim gate can be decomposed in the form of Figure 1.6, and the new matrix, Equation 5.2, is still in this form, just with different single-qubit gates. In this sense, the absorbed SWAP has been performed with *no cost*.

Figure 5.2c shows a layout synthesis solution by OLSQ-GA, the tool to be presented in this chapter, which explores SWAP absorption automatically as part of the layout synthesis. It makes use of 6 absorbed SWAPs and no explicit SWAPs. The achieved depth is 5, which is better than the post-processing solution shown in Figure 5.2b.

There have been a limited number of works taking advantage of SWAP absorption. [ZW19] extended the A* search with SWAP absorption by making the cost of a SWAP to be 0 if it is immediately after a $U(4)$ gate. The most relevant work is a layout synthesis tool from IBM [NBG22, JJB21], which formulates the problem in binary integer programming (BIP) and solves it using a proprietary solver, CPLEX. For the example in [JJB21], OLSQ-GA finds an optimal solution with the same quality (depth 11, 8 SWAPs). On this very example, the solution of [ZW19] has depth 15 and 11 SWAPs. (This is the best case out of 10 trials since its A* algorithm has some randomness.) In comparison, the solution of SABRE has depth 12 and 9 SWAPs [JJB21].

For QAOA and chemical simulation, there is a known optimal layout synthesis solution to the instances with "all-to-all" interactions, like Figure 5.1. In this solution, the gates are arranged in alternating matchings, and each with an absorbed SWAP gate. This corresponds to the most general kind of chemical simulation program, where every qubit has a gate with every other qubit, so the optimal solution has depth $n-1$ with a total of $\binom{n}{2}$ two-qubit gates. [KMW18] provides more details on this optimal layout synthesis solution. Such solution also works for QAOA for complete graphs, i.e., the Sherrington-Kirkpatrick model [HSN21]. However, for problems with fewer gates than the all-to-all interactions, we may not

need $\binom{n}{2}$ gates. In this case, the depth-optimal solution is less structured, and OLSQ-GA is helpful to find it.

5.2 Formulation of OLSQ-GA

In this section, we present optimal layout synthesizer for quantum computing with gate absorption, OLSQ-GA, which formulates the layout synthesis with SWAP absorption into an SMT model [dB08]. There are two inputs to the program as in Figure 5.1: the quantum program consisting of two-qubit gates to map like shown in Figure 5.1a, and the coupling graph of the architecture like shown in Figure 5.1b. The objective of OLSQ-GA is to find a solution with optimal depth or SWAP count as expressed in the following subsection. It is also possible to set the objective to other quantities built from the variables.

5.2.1 Variables

There are 4 groups of variables in OLSQ-GA: mapping, spacetime coordinates, absorbed SWAP, and explicit SWAP. The total number of variables is $|Q|T + 2|G| + 2|E|T$, where $|Q|$ is the number of qubits in the program, T is the number of time steps, $|G|$ is the number of gates, and $|E|$ is the number of edges in the coupling graph. We use q to represent program qubits, p for physical qubits, and e for edges in the coupling graph. We shall use the example in Figure 5.2c for illustration throughout this section.

The mapping variables $\pi_{q,t} = p$ means that, at time t, program qubit q is mapped to physical qubit p, e.g., $\pi_{q_0,0} = p_0$ and $\pi_{q_1,0} = p_3$.

The spacetime coordinates of gate g, $(t_g, x_g) = (t, e)$ means that g is scheduled at time t and locates on edge e in the coupling graph, e.g., the spacetime coordinates for g_0 is $(3, e_2)$ where $e_2 = (p_2, p_3)$.

A set of absorbed SWAP binary variables $\alpha_{e,t}$'s are introduced. If $\alpha_{e,t} = 1$, then there is

an absorbed SWAP on edge e at time t, e.g., $\alpha_{e_3,0} = 1$ since there is a SWAP absorbed by g_5 on edge $e_3 = (p_3, p_4)$ at time 0.

Similarly, a set of explicit SWAP binary variables $\sigma_{e,t}$'s are introduced. $\sigma_{e,t} = 1$ if and only if there is an explicit SWAP on edge e at time t. There is no explicit SWAP in Figure 5.2c, but in Figure 5.2b, $\sigma_{e_1,1} = 1$ since there is a SWAP on edge $e_1 = (p_1, p_2)$ at time 1.

With these variables, the optimization objectives can be easily expressed. Depth is defined as the largest time coordinate of any gate, $T = \max_g t_g$; SWAP count is the sum of all explicit SWAP variables, $S = \sum_{e,t} \sigma_{e,t}$; an estimation of fidelity can be the product of a decoherence factor with all gate fidelity

$$f = e^{-\frac{|Q|\cdot T - 2(|G|+S)}{|Q|\cdot T_0}} f_U^{|G|+S}, \tag{5.3}$$

where $|Q|$ is the number of program qubits, T is the depth, $|G|$ is the number of gates, S is the SWAP count, and T_0 and f_U are hardware factors. T_0 is the decoherence time of a qubit divided by the duration of a $U(4)$ gate, and f_U is the fidelity of a $U(4)$ gate. In physics, decoherence is characterized by an exponential decay with respect to time. So, in Equation 5.3, on the power of the e is the negation of the ratio between the total idle time and the total coherence time.

5.2.2 Constraints

There are five sets of constraints: dependencies, mapping implied by spacetime coordinates, no overlaps, SWAP absorption, and mapping transformation.

Dependencies: For example, $t_{g_4} > t_{g_0}$ and $t_{g_4} > t_{g_1}$. However, if there is a region in the quantum program where all the gates commute with each other, we can simply change the larger-than relation $>$ to non-equality $\neq$. Since the simulation gates commute, the actual constraints are $t_{g_4} \neq t_{g_0}$ and $t_{g_4} \neq t_{g_1}$.

Mapping implied by spacetime coordinates: when gate g acts on program qubit (q, q') at

time t on edge $e = (p, p')$,

$$(t_g == t \ \wedge \ x_g == e) \ \Rightarrow [(\pi_{q,t} == p \ \wedge \ \pi_{q',t} == p') \vee (\pi_{q,t} == p' \ \wedge \ \pi_{q',t} == p)] \,. \quad (5.4)$$

The left-hand side checks the spacetime coordinates of the gate (t, e), while the right-hand side means that, at this time, the mapping of q and q' must be the two physical qubits on e, e.g., since g_8 is at time 0 and on edge $e_1 = (p_1, p_2)$, its two program qubits q_2 and q_4 should be mapped to p_1 and p_2. When we specify an edge with two vertices, there are two possibilities $\pi_{q_2,0} = p_1$ and $\pi_{q_4,0} = p_2$, or $\pi_{q_2,0} = p_2$ and $\pi_{q_4,0} = p_1$, which is how the right hand side of Equation 5.4 got its form. In our example, the former case is true.

No overlaps: there are only two types of gates in our layout synthesis solution, $U(4)$ gates from the program and the explicit SWAPs. The $U(4)$ gates cannot overlap with each other by the dependency constraints, so we only need to consider the overlaps between $U(4)$ gates and SWAPs, and among SWAPs themselves. For two incident edges e and e', any gate g, and any time t,

$$\sigma_{e,t} == 1 \quad \Rightarrow \quad \sigma_{e',t} == 0, \quad (5.5)$$

$$(t_g == t \ \wedge \ x_g == e) \quad \Rightarrow \quad \sigma_{e',t} == 0, \quad (5.6)$$

e.g., there can be no explicit SWAP on edge $e_0 = (p_0, p_1)$ or $e_2 = (p_2, p_3)$ at time 0 since there is a gate g_8 on an incident edge $e_1 = (p_1, p_2)$ at time 0. In Figure 5.2b, there is a SWAP scheduled at time 1 on e_1, so there cannot be any other SWAPs or $U(4)$ gates on incident edges e_0 or e_2 at time 1.

SWAP absorption: without constraints, an absorbed SWAP can happen on any edge at any time, which is clearly not possible. If there is an absorbed SWAP on edge e at time t, there should also be some $U(4)$ gate, i.e., for any time t and edge e,

$$\alpha_{e,t} == 1 \quad \Rightarrow \quad \bigvee_g (t_g == t \ \wedge \ x_g == e) \,, \quad (5.7)$$

e.g., if there is an absorbed SWAP on $e_3 = (p_3, p_4)$ at time 0, then there must be a gate (g_5 in our example) having spacetime coordinates $(0, e_3)$.

Mapping transformation: there are two sources of change for the layout synthesis solution: absorbed and explicit SWAPs. If either one of them is 1, we deduce the new mapping from the old mapping, i.e., for any qubit q, any time t, and any edge $e = (p, p')$,

$$\begin{aligned} [\pi_{q,t} == p \ \wedge \ (\sigma_{e,t} == 1 \ \vee \ \alpha_{e,t} == 1)] \quad &\Rightarrow \quad \pi_{q,t+1} == p', \\ [\pi_{q,t} == p' \ \wedge \ (\sigma_{e,t} == 1 \ \vee \ \alpha_{e,t} == 1)] \quad &\Rightarrow \quad \pi_{q,t+1} == p, \end{aligned} \tag{5.8}$$

e.g., q_1 is mapped to p_3 at time 0, but there is an absorbed SWAP on edge $e_3 = (p_3, p_4)$. As a result, at time 1, q_1 is mapped to p_4. Similarly, the mapping of q_3 changes from p_4 to p_3 at time 1. Thus, g_9 acting on q_3 and q_4 can be executed at time 1 on $e_2 = (p_2, p_3)$, but not at time 0 because of the mapping. On the other hand, if there are no SWAPs, absorbed or explicit, on any edge going into the current physical qubit, the mapping remains unchanged from t to $t + 1$, i.e., for any qubit q, any time t, and any physical qubit p,

$$\left[\pi_{q,t} == p \ \wedge \left(\sum_{e \in E_p} \sigma_{e,t} == 0\right) \wedge \left(\sum_{e \in E_p} \alpha_{e,t} == 0\right)\right] \quad \Rightarrow \quad \pi_{q,t+1} == p, \tag{5.9}$$

where E_p is the set of edges incident on physical qubit p. For instance, at time 1, there is neither absorbed nor explicit SWAP on e_0 or e_1, so the mapping of q_2 remains at p_1.

5.3 Evaluation of OLSQ-GA

QAOA can be adapted to many optimization problems. One of the promising candidates is the MAX-CUT problem on 3-regular graphs [HWO19]. A QAOA program typically consists of p iterations ($p \in \mathbb{N}$), and each iteration consists of two stages: *phase-splitting* and *mixing*. The mixing stage only has single-qubit gates, so there is no layout synthesis issue. The phase-splitting stage, however, presents an interesting layout synthesis problem. Specifically, for the MAX-CUT problem on a graph $G = (V, E)$, each qubit encodes a vertex, and we need to apply a two-qubit gate on every edge of G. These gates are all commutable. A state-of-the-art experimental work at the time used a heuristic compiler [SDC20] and coupling graph in Figure 5.3, but the quality of the result dropped quickly with increasing problem sizes.

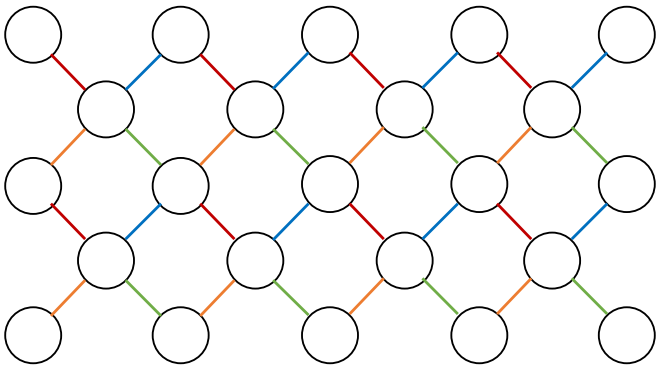


Figure 5.3: Part of the Google Sycamore architecture [HSN21]. Four different colors represent four maximal matchings of the coupling graph.

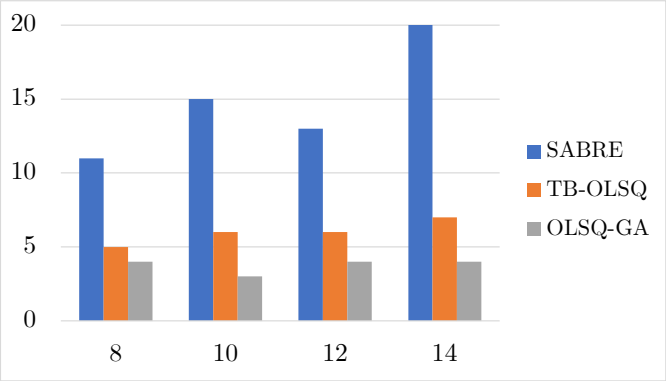


Figure 5.4: Depth results by SABRE, TB-OLSQ, and OLSQ-GA.

We implemented OLSQ-GA[1] with Z3 SMT solver [dB08] and generated four 3-regular graphs of sizes 8, 10, 12, and 14 with the NetworkX package [HSS08] as the benchmark, similar to the setting in Google's experimental work [HSN21]. We evaluated OLSQ-GA against two tools with the same benchmark: SABRE [LDX19], and TB-OLSQ [TC20]. Although SABRE is not exactly what was used in [HSN21], it is also considered to be a state-of-the-art for heuristic layout synthesis [JJB21]. TB-OLSQ uses an optimal approach but does not take the gate absorption into consideration. We set the number of SWAPs as the objective in OLSQ-GA. The depth, SWAP count, and fidelity of the layout synthesis solutions for the

[1] https://github.com/UCLA-VAST/OLSQ/tree/GateAbsorption

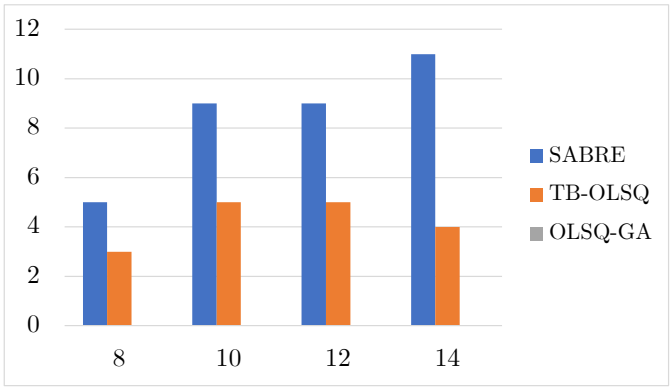


Figure 5.5: SWAP count results by SABRE, TB-OLSQ, and OLSQ-GA. Note that OLSQ-GA managed to insert no explicit SWAP gates, so there are no gray bars in the graph above.

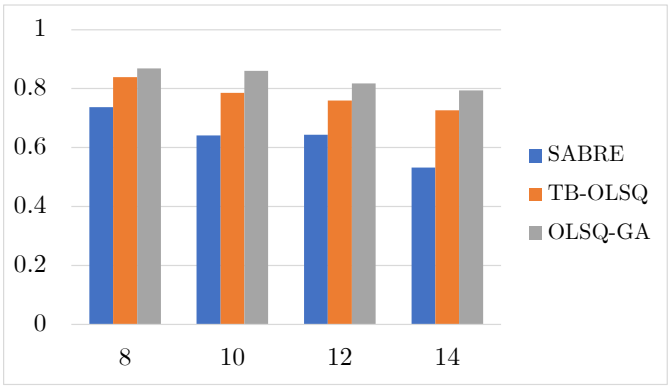


Figure 5.6: Fidelity results by SABRE, TB-OLSQ, and OLSQ-GA.

phase-splitting stage of a single iteration in QAOA are shown in Figure 5.4, Figure 5.5, and Figure 5.6 respectively. The fidelity is estimated by Equation 5.3 with slightly optimistic parameters $T_0 = 50$ and $f_U = 0.99$, which means that decoherence time is 50x the $U(4)$ gate duration, and each $U(4)$ gate fidelity is 99%. As we can see, the heuristic tool, without consideration of the SWAP absorption or commutation, returns solutions with the highest depth and SWAP counts, and thus the lowest fidelity. The results of TB-OLSQ are already significantly better than the heuristic results. OLSQ-GA performs the best of all three. Compared to TB-OLSQ, it reduces depth by up to 50.0% and SWAP count by 100% while improving fidelity by 9.45%. Compared to SABRE, it reduces depth by up to 80.0%, SWAPs

by 100% and improves fidelity by up to 49.1%.

Note that all the layout synthesis solutions for one iteration can easily extend to multiple iterations: we can simply reverse the order of all the gates and append this reversed circuit as the second iteration. Of course, in the new iteration, there are different parameters in the gates, but the layout synthesis problem can be solved just for one iteration. This way, the final mapping of all the odd iterations is the same as the final mapping of the first iteration, and the final mapping of the even iterations is just the initial mapping. The total fidelity of all iterations is the product of fidelity of each, so the total fidelity would be exponential to the single-iteration fidelity. Since the QAOA circuits with more iterations contain the QAOA circuits with less iterations, in the ideal case, the quality of QAOA results should increase as the number of iteration p increases. However, in the leading experimental work at the time [HSN21], such a trend is only observed on hardware-native graphs, not the generated 3-regular graphs like what we use in this chapter. Without quantum error correction, the fidelity of the whole circuit is only going to decrease as the number of gates increases. However, the quality of the QAOA results is not proportional to the circuit fidelity, which is why some improvements are still observed.

Apart from still-low gate fidelity, we believe one of the reasons is the sub-optimal compilation for layout synthesis, e.g., authors of [HSN21] report that the depth of the heuristic mapping solutions for a single QAOA iteration is approximately the size of the 3-regular graph. In comparison, the depth of OLSQ-GA results stays as a constant (3 or 4), which is way less than the size of the graphs (8 to 14), and the same or lower than the hardware-native graphs. This suggests that, using OLSQ-GA, the existing hardware capability could also demonstrate improvements with more iterations for 3-regular graphs. Figure 5.7 shows the fidelity of three mapping approaches with up to 5 iterations. As the number of iterations increases, the advantage of OLSQ-GA becomes more visible: compared to TB-OLSQ, it gains fidelity by 30.5% for 3 iterations, and 55.9% for 5 iterations; compared to SABRE, the improvements are 231% are 636%.

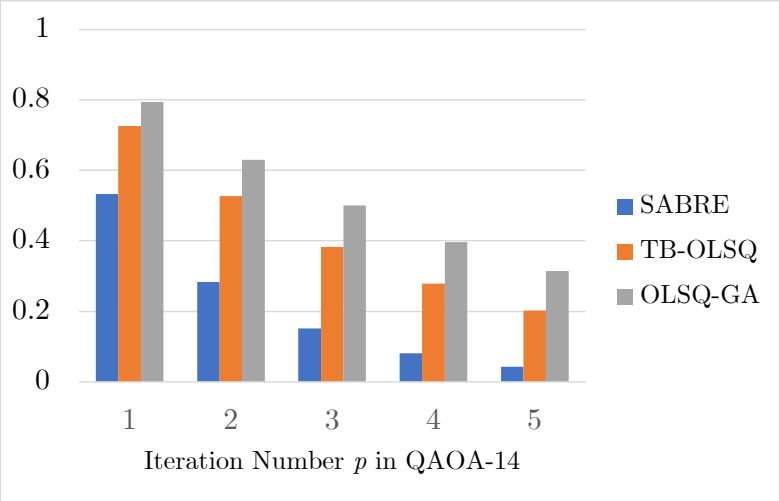


Figure 5.7: Fidelity of multiple iterations of QAOA-14 (QAOA for the 3-regular graph with 14 vertices) by SABRE, TB-OLSQ, and OLSQ-GA.

5.4 Solution Space Reduction

The solution space of SMT models for OLSQ-GA may be reduced with adding more constraints, thus speeding up the solver. We shall present two techniques in this section: using alternating matchings pattern and setting initial mapping.

5.4.1 Analysis on Optimal Mapping Solutions

In graph theory, matching is a set of pair-wise nonadjacent edges, none of which are self-loops. The $U(4)$ gates executing at the same time step t consist of a matching on the coupling graph, M_t. In general, we have the following theorem.

Theorem 2. *In a depth-optimal layout synthesis solution, for any t, $M_t \cup M_{t+1}$ cannot be a matching for the coupling graph.*

Proof: If $M_t \cup M_{t+1}$ is a matching in a depth-optimal layout synthesis solution S, we can move all of $M_{t+1} \setminus M_t$ to time t, and absorb all of $M_{t+1} \cap M_t$ to the corresponding gates in M_t. (Note that, due to the generality of $U(4)$, not only SWAPs, but also any $U(4)$ gate can

be absorbed into another $U(4)$ gate.) Then we get a new layout synthesis solution S' where the time step $t+1$ is not needed, which contradicts the fact that S is depth-optimal. □

For example, in Figure 5.2b, $M_1 = M_2 = \{(p_1, p_2), (p_3, p_4)\}$, so $M_1 \cup M_2 = M_1 = M_2$ is a matching of the coupling graph, Figure 5.1b. The gates at time 2 can be absorbed into gates at time 1.

Analyzing the solution in Figure 5.2c, we can observe a pattern: the gates alternate between two matchings $M_0 = \{(p_1, p_2), (p_3, p_4)\}$ and $M_1 = \{(p_0, p_1), (p_2, p_3)\}$. In fact, for linear architecture, we can formalize this observation. We call the edges in M_0 even edges, and edges in M_1 odd edges.

Corollary 2.1. *For the layout synthesis problem of programs with commutation to a linear architecture with coupling graph $G = (P, E)$ where P is the set of physical qubits, and $E = \{(p_i, p_{i+1}) | i = 0, ..., |P| - 2\}$, there is always an depth-optimal layout synthesis solution such that the time steps alternate between sets of even edges and sets of odd edges.*

Proof: From Theorem 2, $M_t \cup M_{t-1}$ cannot be a matching. For the linear architecture, this means that there are both odd and even edges in $M_t \cup M_{t-1}$, since with only odd or even edges, $M_t \cup M_{t-1}$ would still be a matching. By absorbing and moving, we can always build new time steps $t-1$ and t such that $t-1$ only has gates on even edges, and t only has gate on odd edges. Since $M_t \cup M_{t-1}$ has both even and odd edges, none of the two new time steps can be empty. As a result, we have constructed a new optimal solution satisfying the alternating matchings pattern with the same depth. □

5.4.2 Implementing Alternating Matchings Pattern

For linear architecture, Corollary 2.1 leads to a great reduction in solution space of the layout synthesis problem without loss of optimality. In the OLSQ-GA formulation, this can be achieved by assigning values to many explicit SWAP and space variables for $U(4)$ gates.

Table 5.1: OLSQ-GA speedup with extra constraints. The architectures for simulation and QV64 are linear. The architecture for QAOA is shown in Figure 5.3. Baseline is the runtime in seconds. 'init.' means fixing initial mapping. 'match.' means using alternating matchings pattern. The asterisk (*) means that the mapping solution using the corresponding technique(s) may not be optimal. However, the depth of the two cases with data shown above matched the depth certificate as in Section 5.4.3, so these solutions are indeed optimal.

problem	**objective**	**baseline**	**match.**	**init.**	**both**
5-qubit simulation	SWAP	4.74E0	1.40x	2.58x	2.44x
QV64	Depth	2.40E2	6.35x	5.00x	8.86x
	SWAP	8.50E3	95.4x	53.0x	272x
QAOA-14, Sycamore	Depth	1.65E5	8.41x*	522x*	*

For all (t, e_k) such that $t - k = 1 \mod 2$, and all gate g,

$$\sigma_{e_k,t} = 0, \tag{5.10}$$

$$t_g == t \quad \Rightarrow \quad x_g \neq e_k. \tag{5.11}$$

These constraints make sure that there are only gates on even edges at time steps 0, 2, 4, ... And there are only gates on odd edges at time steps 1, 3, 5, ... If there is an even number of edges in the linear architecture, these constraints suffice. However, if there is an odd number of edges, we may need to try another case with $t - k = 0 \mod 2$ instead of 1. The two matchings have a different number of edges, so it matters which one we start from. Taking the better result of the two cases, we derive the optimal result.

Since we have fixed some variables and added more constraints, the solution space for the SMT solver to explore is smaller, which results in a faster runtime. Some speedup results are shown in Table 5.1. For layout synthesis of the QV64 circuit [JJB21], alternating matchings bring 95.4x speedup.

For generic architectures, it is more complex. For example, for a 2D architecture like Figure 5.3, there are four maximal matchings, shown in different colors, which are mutually disjoint in a sense similar to M_0 and M_1. [HSN21] alternates among these matchings to schedule a single QAOA iteration for hardware-native graphs: at each time step, a group of gates with the same color are executed. However, the optimal layout synthesis solution for other quantum circuits may use other possible ordering of these four matchings, or even other possible matchings.

5.4.3 Depth Certificate

There is a generic case where we can guarantee optimal depth even with heuristics: we can run two OLSQ-GA instances with the heuristics turned on and off. The two instances start with a certain maximal depth. If the current maximal depth is too low to yield any solution, OLSQ-GA would increase the maximal depth and start over. The exact instance explores a larger solution space, so its runtime is longer. Meanwhile, it can output what is the maximal depth currently being explored, e.g., 4, which serves as a certificate that no solutions with depth less than 4 can be found. Now, if the heuristic instance returns a mapping solution with depth 4, which takes less time than the exact instance, then the solution is optimal because of the depth certificate by the exact instance. In Table 5.1, we also report the speedup of the 14-qubit QAOA with alternating matchings, using depth as the objective. For the SWAP count, the optimality argument would be harder to guarantee. However, if the heuristic solution does not contain any explicit SWAPs, then it is optimal with respect to SWAP count.

5.4.4 Setting Initial Mapping

Another technique to reduce solution space is to set initial mapping. If there are not too many qubits, we can send instances with different initial mappings to different cores and

perform the solving in parallel. For problems with a strong symmetry, we can set initial mapping to "break" some symmetry without loss of optimality, e.g., in the 5-qubit all-to-all chemical simulation, we can use arbitrary initial mapping, and the speedup is 2.58x. As implementation, we can add these constraints to OLSQ-GA:

$$\pi_{q_i,0} = p_i \quad \text{for } i = 0, ..., 4 \tag{5.12}$$

If the gates are not commutable like in QV64, the initial mapping should enable some gates to execute since the SWAPs before all the $U(4)$ gates can simply be left out and we set the initial mapping to be directly whatever mapping it is after these "prelude" SWAPs. QV also has a special property that its first time step is a maximal matching consisting of $\lfloor n/2 \rfloor$ gates. Being exhaustive, we can let each core in a computational cluster try one of the $\lfloor n/2 \rfloor!\ 2^n/2$ possible initial mappings. The factorial term is the number of mappings from the gates to edges on the coupling graph. The exponential term is for both directions of each edge. Finally, note that, if the architecture is 1D and we reflect a mapping solution with respect to the center, we get another solution with the same depth and SWAP count. Thus, we can divide the possibilities of initial mapping by 2 in Equation 5.12. Using Sterling's approximation, the asymptotic of this value is $\sqrt{\pi n}(n/e)^n/2$, which is approximately 35% of all the possible initial mappings $n!$. For $n = 6$, the required core count is 192, which is not too much in distributed computing. The best solution of all these cases is still guaranteed to be optimal. We chose one possibility and achieved 53x speedup, as shown in Table 5.1. With both alternating matching and initial mapping, we achieve up to 272x speedup.

We can also use the initial mapping results as the heuristic in Section 5.4.3. For example, we used TB-OLSQ to derive an initial mapping for the 14-qubit QAOA and use it in OLSQ-GA. The combined runtime of TB-OLSQ and OLSQ-GA is still 522x faster than the baseline. However, note that combining alternating matchings and initial mapping may cause issues. The initial mapper may not produce an alternating matchings solution. So, it cannot be combined with alternating matchings to produce a depth-optimal solution.

CHAPTER 6

OLSQ-DPQA: Layout Synthesis for Dynamically Field-Programmable Qubit Arrays Based on Satisfiability Modulo Theories

From this chapter, we embark on a journey of layout synthesis for the dynamically field-programmable qubit arrays architecture (DPQA), which has dynamic connectivity among qubits. As a first step, we summarize the physics of the architecture into several implications. Then, we present an SMT formulation that generates layout synthesis solutions satisfying all the implications.

6.1 The DPQA Architecture

In DPQA, each qubit is a single atom trapped in an individual optical tweezer, which enables a deterministic control over the qubit position. The physics of atomic trapping, optical tweezers, and entangling gates leads to several key implications. These implications serve as the interface between physics and computer science where we reason about variables, constraints, and optimization procedures. Thus, we enumerate the implications for reference. For the specific parameters, we follow the leading experimental work [EBK23, BLS22].

6.1.1 Atom Trapping

One trap cannot hold more than one atom. Otherwise, the atoms may expel each other out of the trap.

Implication 1. One trap can hold zero or one atom at any time during the computation.

Two orthogonal optical components generate AOD (acousto-optic deflector) tweezers. The X component produces a horizontal pattern, and the Y component multiplies this pattern by a vertical pattern. In contrast, an arbitrary phase hologram on a spatial light modulator produces SLM tweezers. As a result, we can place each SLM tweezer in an arbitrary location. However, to enable massive parallelism of gate execution, the geometry of the SLM and the AOD should be similar.

Implication 2. AOD and SLM optical trap arrays are rectangular arrays that extend in the X and Y direction in the 2D plane.

For example, in Figure 6.4b, the AOD is a rectangular array with two rows and four columns, indicated by the dashed grid. The dynamically programmable processor in [BLS22] uses up to 24 qubits, but system sizes of 100s of qubits are attainable as was done in [EKC22], and both SLM and AOD grids have been used in system sizes as large as 16x16 each [SAP22].

Because of the finite optical resolution of the microscope generating tweezers, traps of the same array cannot be closer than a given minimum spacing. In [BLS22], it is 2 um.

Implication 3. There is a minimal separation between two rows or columns of traps in the same array, d_s.

6.1.2 Array Movements

AOD traps can move whereas SLM traps cannot. Thus, it may seem to some readers that SLM is strictly less general than AOD, rendering the notion of SLM redundant for

compilation. However, an advantage of SLM is that we can turn off the unused traps based on the compilation result. As part of the architecture specification, we make a certain number of SLM traps available to the compiler, but some of them are never used throughout the compiled result. Then, we simply ignore them when we generate the SLM in the beginning of the experiment, which saves some laser power. Although total laser power is not a bottleneck at the moment, the savings of SLM are beneficial for future scaling-up. Thus, we keep SLM in the formulation instead of just treating it as a special case of AOD.

Implication 4. If the array is the SLM type, the traps are stationary.

For example, q_4 stays at the same place throughout Figure 6.4.

The control we have on the AOD traps are the Y coordinate of each row and the X coordinate of each column.

Implication 5. If the array is the AOD type, a row/column of traps move together.

For example, from Figure 6.4b to 6.4c, the AOD row of q_5, q_3, and q_1 moves upwards, and the column of q_2 and q_3 moves to the right.

Per Implication 3, we cannot place two rows/columns too close together. If rows A and B move across each other, they must have been closer than the minimum spacing at some point, which is prohibited.

Implication 6. If the array is the AOD type, a row cannot cross over another row, a column cannot move over another column.

In [BLS22], the relation between movement time t and travel distance D is set as $t = T_0\sqrt{D/D_0}$ to maintain constant heating of the atoms during movements. We follow their setting T_0 =200 us and D_0 =110 um so that the heating is sufficiently low.

6.1.3 Quantum Gates

Single-qubit gates are high-fidelity operations that are generically easy to perform locally (see [LKS19]).

Implication 7. Arbitrary single-qubit gates can be addressed to each qubit individually.

We perform two-qubit operations with a specific type of laser to excite the atoms to Rydberg state. In this state, atoms within a certain distance will interact strongly and cannot be excited simultaneously. The characteristic distance of this interaction is the Rydberg blockade radius r_b (7.5 um in [BLS22]). This blockade mechanism is the basis of two-qubit entangling gates; only if two atoms are within a r_b of each other can they perform a two-qubit gate. The specific gate implemented in [BLS22] is the Levine-Pichler CZ gate, which is a special case of controlled-R_z gates available in DPQA [LKS19].

Implication 8. Two qubits q and q' can only perform an entangling two-qubit gate when they are within a blockade radius, i.e., $|\vec{x}_q - \vec{x}_{q'}| \leq r_b$, and they are both illuminated by the Rydberg laser.

The Rydberg laser is *global* in the sense that it illuminates *all the qubits*, as done in [EKC22, BLS22]. When we turn on the laser, we cannot "switch off" the interaction of a pair if they are within range.

Implication 9. If q and q' are within r_b and illuminated by the Rydberg laser, they will go through a two-qubit entangling gate.

If two atoms are sufficiently separated, $> 2.5r_b$ in practice, they will not interact even if excited by the Rydberg laser. If there are more than two atoms that are not sufficiently separated, they go through a joint quantum process which is not a well-defined gate.

Implication 10. For any three qubits q_0, q_1, and q_2, at most one of the following is true when the Rydberg laser is on: $|\vec{x}_{q_0} - \vec{x}_{q_1}| < 2.5r_b$, $|\vec{x}_{q_1} - \vec{x}_{q_2}| < 2.5r_b$, and $|\vec{x}_{q_2} - \vec{x}_{q_0}| < 2.5r_b$. That is, only disjoint pairs of qubits may entangle simultaneously.

6.1.4 Error Source

Errors can occur during the gates or the idling time (including AOD movements, activation, and deactivation). In the evaluation in this chapter, the average idling time is only 2.6% of the qubit lifetime (coherence time) in the largest benchmark (90-node QAOA). Thus, the idling time plays a relatively small role in the error source. In addition, the single-qubit operations are significantly higher fidelity (99.99%) than the two-qubit entangling gates (99.5%) [EBK23]. A global Rydberg laser for the two-qubit gates induces the same error rate on all qubits whether they are involved with a two-qubit gate at this stage or not.

Implication 11. The main computational error source is the number of layers of two-qubit gates.

6.1.5 Atom Transfer

So far, we have described atoms staying in their own individual tweezer traps, as was focused on in the experiments of [BLS22]. However, it has previously been demonstrated that atoms can be transferred between tweezer traps [BTM07] by reducing the intensity of one tweezer trap while increasing or maintaining the intensity of another tweezer trap. In the system considered here, in an AOD array, we can tune the individual intensity of AOD rows and columns to transfer to/from SLM traps: e.g., Figure 6.4e, we turn off the leftmost AOD column so that q_5 is transferred to SLM.

6.1.6 Universality

With atom transfers, the architecture can perform universal quantum computing given a large enough area. Figure 6.1 depicts a toy construction. We load the qubits to one SLM row with sufficient separations between the traps. There is one AOD trap working as delivery. Per Implication 7, single-qubit gates are always executable. To apply an entangling gate on an arbitrary pair (q_i, q_j), we perform the 4-step procedure illustrated in Figure 6.1. Finally,

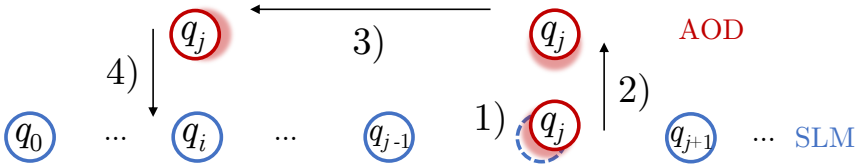


Figure 6.1: Universal quantum computing on DPQA with one AOD trap and one SLM row. Single-qubit gates execute directly. To implement an entangling gate on an arbitrary qubit pair: 1) pick up q_j from SLM to AOD, so the original SLM trap is empty (dashes), 2) shift the AOD trap up, 3) move the AOD trap horizontally until q_i and q_j align, then 4) shift the AOD trap down to perform the gate.

we reverse the movements and put q_j back to SLM. Now, we are ready for the next gate. With this construction, we can execute any single-qubit and two-qubit gate, so the architecture can perform universal QC. Of course, this construction is like the demonstration of the Turing machine in classical computing where efficiency is not considered. For example, we can easily put atoms in a square array that reduces the amount of time for movements.

6.2 Discretization of the Solution Space

As pointed out previously, we have the freedom to specify the locations of an AOD row r as a function of time $y_r(t)$ and, similarly, $x_c(t)$ of an AOD column c. Modeling the DPQA architecture based on these continuous functions is cumbersome and unnecessary for a compiler. In fact, the time domain can be easily discretized to Rydberg *stages* like in Figure 1.5b because we only care about the location of qubits when we turn on the Rydberg laser to apply the entangling gates. As long as we do not violate the DPQA constraints, the 2D planar movements of AOD between any two stages can be straightforwardly interpolated. We can implement single-qubit gates using individually addressable lasers between stages, so we filter out the two-qubit gates and compile them. After this compilation, we can reintroduce single-qubit gates. For more details on this, kindly refer to Section 6.5.

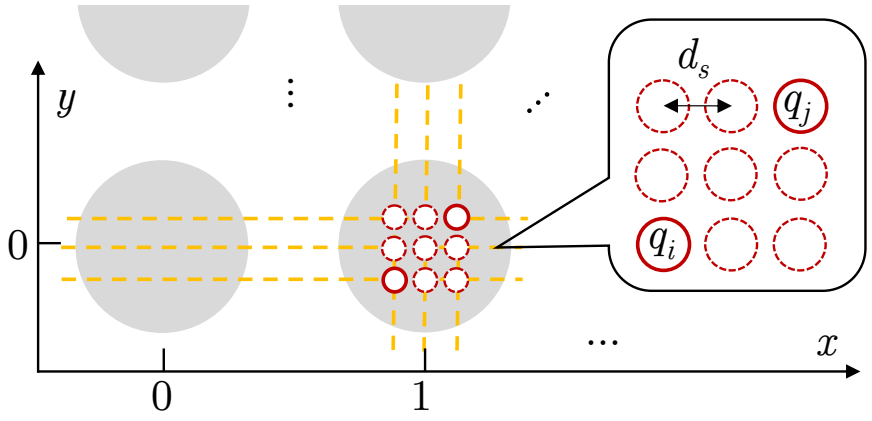


Figure 6.2: Discretization of space into interaction sites. The unit of X and Y is a sufficient distance to prevent Rydberg interaction. Interaction sites, indicated by shades, are centered at integer points on the 2D plane. A limited number of AOD rows or columns can stack together at one site. The callout is zooming into a site with three AOD rows and three columns.

There may be parallel executions of two-qubit entangling gates at different sites, so, per Implication 10, the sites should be sufficiently separated to avoid unwanted Rydberg interactions. Also, to maximize usage, the tiling pattern of the sites should accord to the geometry of the tweezer arrays, which is a 2D grid per Implication 2. The interaction sites are illustrated as shades in Figure 6.2. In fact, our effort in discretization is analogous to that of Mead and Conway [MC80] in VLSI chip design where an abstract basic length unit in semiconductor fabrication, λ, was introduced. The chip area is discretized to separated "lines" of 2λ wide layout design. These dimensionless λ-rules helped the advancement of automated layout tools despite the fast developments in the fabrication technology that affects λ. Similarly, based on our discretization, our formulation holds even if the constants r_b, $2.5r_b$ and d_s change. It is crucial to retain this flexibility for possible adjustments in physics experiments. For instance, we may want to excite the qubits to a higher Rydberg state, leading to a bigger r_b; or upgrading to higher-resolution microscope objective lenses, leading to a smaller d_s.

We allow several rows or columns to "stack" together at one interaction site to support gates between two AOD qubits. However, there is an upper bound on how many AOD rows/columns can be stacked together at a site because these AOD rows/columns cannot be too close to each other (Implication 3). We denote the maximal stacking factor as R_{STK} and C_{STK}, respectively. They are decided by the minimal AOD row/column separation d_s and the Rydberg range r_b. The callout in Figure 6.2 exhibits an extreme case where we need to entangle two qubits q_i and q_j at opposite corners of the site. This requires $[(R_{\text{STK}} - 1)^2 + (C_{\text{STK}} - 1)^2]d_s^2 \leq r_b^2$. With r_b =7.5 um and d_s =2 um, $R_{\text{STK}} = C_{\text{STK}} = 3$ satisfies the inequality.

In fact, the ticks on x and y axes in Figure 1.5b and Figure 6.4b-f indicate the interaction sites. At each stage, per Implication 10, there can be at most two qubits. Thus, there are five possible situations at a site: 1) empty, e.g., (0,1) at stage 0 (Figure 6.4b); 2) one SLM qubit, e.g., (1,1) at stage 2 (Figure 6.4e) holds only q_4; 3) one AOD qubit, e.g., (3,1) at stage 0 holds only q_0; 4) one SLM qubit and one AOD qubit, e.g., (1,1) at stage 0 holds q_4 and q_2; and 5) two AOD qubits, e.g., (1,0) at stage 0 holds q_5 and q_3.

The discretized coordinates (of interactions sites) are enough to specify AOD and SLM qubit locations, but they are not sufficient as the state of the architecture because of the stacking of rows/columns we just mentioned. For example, at stage 1 (Figure 6.4c), both AOD rows are at $y = 1$. Because of Implication 6, the upper row cannot move across the lower row, e.g., q_2 cannot move below q_3. With only coordinates, it is hard to enforce constraints like this. Thus, as part of the architecture state, we also need to specify which row and column each AOD qubit is in. Finally, we have to specify whether the qubit is in SLM or AOD at each stage to handle atom transfers.

In conclusion, the computation progresses in multiple stages: stage 0, AOD movement 0, stage 1, AOD movement 1, stage 2, ... At each stage, the architecture has a state consisting of interaction site indices (specifying location), AOD row/column indices, and an array index (specifying whether in SLM or AOD) for each qubit. During the AOD movement, the AOD

row/column indices and the array index are invariant, but the site indices can change as AOD traps move in space.

6.3 Optimal Compilation with SMT

With the discretization in the previous section, we can use variables and constraints to encode the layout synthesis problem and then, we can invoke an SMT solver for optimal solutions, as illustrated in Figure 6.3. We shall work through the example of compiling the quantum circuit in Figure 6.4a to DPQA to explain the formulation.

6.3.1 Variables

Site indices $x_{i,s}$ and $y_{i,s}$: at stage s, qubit q_i is at interaction site $(x_{i,s}, y_{i,s})$, e.g., for q_0 at stage 0 (Figure 6.4b), $x_{0,0} = 3$ and $y_{0,0} = 1$; at stage 1 (Figure 6.4c), $x_{0,1} = 3$ and $y_{0,1} = 1$ still; at stage 2 (Figure 6.4e), $y_{0,2} = 1$ still, but $x_{0,2} = 2$ due to movements.

Array index $a_{i,s}$: at stage s and the movement following it, if q_i is in SLM, $a_{i,s} = 0$; if it is in AOD, $a_{i,s} = 1$, e.g., for q_5 at stages 0 and 1, $a_{5,0} = a_{5,1} = 1$; before stage 2, it is transferred to SLM, so $a_{5,2} = 0$.

AOD column/row indices $c_{i,s}$ and $r_{i,s}$: at stage s and the movement following it, qubit q_i is at AOD column $c_{i,s}$ and row $r_{i,s}$, e.g., at stage 0, $r_{5,0} = 0$ and $c_{5,0} = 0$ for q_5; $r_{0,0} = 1$ and $c_{0,0} = 3$ for q_0. (We index the row from below and the column from left.) Since it is unknown in advance whether a qubit will be in AOD or SLM, we introduce the r and c variables for all qubits, but only those for AOD qubits will play a role in constraints.

Time coordinate t_j: gate g_j is scheduled to stage t_j, e.g., g_0 in Figure 6.4a is on q_2 and q_4 and at stage 0, so $t_0 = 0$; g_1 is also at stage 0 (Figure 6.4b), so $t_1 = 0$. g_7 and g_8 are at stage 3 (Figure 6.4f), so $t_7 = t_8 = 3$.

We provide the values of all the SMT variables in the running example in Table 6.1.

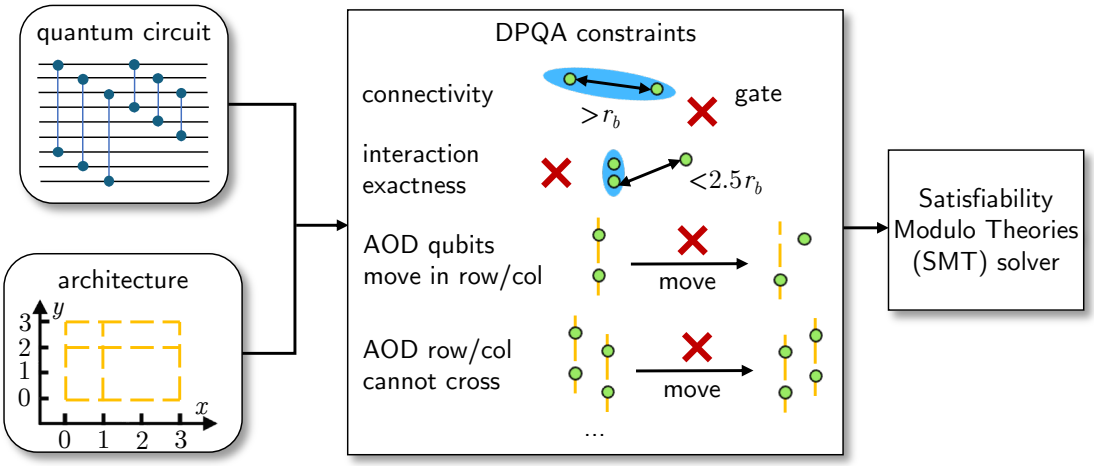


Figure 6.3: Illustration of the OLSQ-DPQA approach. The input consists of the quantum circuit to execute and the DPQA architecture specification, e.g., how large the plane is and how many AOD rows and columns we can have. The compiled instructions have to respect the constraints of DPQA. For example, when a two-qubit gate is executed, the two qubits should be closer than r_b and there cannot be another qubit nearby. Also, all traps in the same AOD row/column move together and must stay in the same order from the beginning to the end of the process. We formulate all the constraints to a satisfiability modulo theories (SMT) model and use an existing SMT solver to find solutions, with which we can derive valid DPQA instructions to run the circuit.

For array indices, we make the values for the last stage gray because they do not affect the solution in any way. (Note that for S stages, there are only $S-1$ movements in between; and our convention is that the movement between s_i and s_{i+1} is encoded in the a, c, and r variables of s_i.) The bounds for site indices and AOD column/row indices are $X=4$, $Y=2$, $C=4$ and $R=2$. Apart from the values for the last stage, some other values are also gray because the corresponding qubit is in SLM in that stage. For convenience in comparing with Figure 6.4, we also reorganize the values based on stages as follows.

```
stage0: [
    {qubit: 0, a: 1, x: 3, y: 1, c: 3, r: 1},
```

```
    {qubit: 1, a: 1, x: 2, y: 0, c: 2, r: 0},
    {qubit: 2, a: 1, x: 1, y: 1, c: 1, r: 1},
    {qubit: 3, a: 1, x: 1, y: 0, c: 1, r: 0},
    {qubit: 4, a: 0, x: 1, y: 1, c: 2, r: 1},
    {qubit: 5, a: 1, x: 1, y: 0, c: 0, r: 0}
];
stage1: [
    {qubit: 0, a: 1, x: 3, y: 1, c: 3, r: 1},
    {qubit: 1, a: 1, x: 3, y: 1, c: 2, r: 0},
    {qubit: 2, a: 1, x: 2, y: 1, c: 1, r: 1},
    {qubit: 3, a: 1, x: 2, y: 1, c: 1, r: 0},
    {qubit: 4, a: 0, x: 1, y: 1, c: 1, r: 0},
    {qubit: 5, a: 1, x: 1, y: 1, c: 0, r: 0}
];
stage2: [
    {qubit: 0, a: 1, x: 2, y: 1, c: 3, r: 1},
    {qubit: 1, a: 1, x: 2, y: 0, c: 2, r: 0},
    {qubit: 2, a: 1, x: 2, y: 1, c: 1, r: 1},
    {qubit: 3, a: 1, x: 2, y: 0, c: 1, r: 0},
    {qubit: 4, a: 0, x: 1, y: 1, c: 1, r: 0},
    {qubit: 5, a: 0, x: 1, y: 0, c: 1, r: 1}
];
stage3: [
    {qubit: 0, a: 1, x: 1, y: 1, c: 3, r: 1},
    {qubit: 1, a: 1, x: 1, y: 0, c: 2, r: 0},
    {qubit: 2, a: 0, x: 0, y: 1, c: 1, r: 1},
    {qubit: 3, a: 1, x: 0, y: 0, c: 1, r: 0},
```

```
    {qubit: 4, a: 0, x: 1, y: 1, c: 2, r: 1},
    {qubit: 5, a: 1, x: 1, y: 0, c: 3, r: 1}
].
```

6.3.2 Constraints

Constraints in this subsection come from physics implications on the architecture (circuit-independent), or fundamental properties of quantum programs (circuit-dependent). Let us use N for the number of qubits and G for the number of gates. Note that in the constraints below, we use '==' to denote the operation that returns Boolean true if the left-hand side equals the right-hand side and returns false otherwise. '$[A, B)$' means from A to $B-1$. All the concrete examples are from Figure 6.4, and the reader can plug in values from Table 6.1 for more examples.

6.3.2.1 Circuit-Independent Constraints

Upper bounding the variables: $\forall i \in [0, N), s \in [0, S)$

$$0 \leq x_{i,s} < X, \tag{6.1}$$

similarly for y, c, and r with bounds Y, C, and R.

Stationary SLM enforces Implication 4: $\forall i \in [0, N),\ \forall s \in [0, S-1)$

$$a_{i,s} == 0 \quad \Rightarrow \quad (x_{i,s+1} == x_{i,s} \ \wedge \ y_{i,s+1} == y_{i,s}). \tag{6.2}$$

For example, q_4 is in SLM at stage 0, i.e., $a_{4,0} = 0$, so its site indices remain the same between stage 0 and 1, i.e., $x_{4,1} = x_{4,0}$ and $y_{4,1} = y_{4,0}$.

AOD moves by whole rows/columns enforcing Implication 5: $\forall i \in [0, N),\ \forall s \in [0, S-1)$

$$a_{i,s} == 1 \quad \Rightarrow \quad (c_{i,s+1} == c_{i,s} \ \wedge \ r_{i,s+1} == r_{i,s}). \tag{6.3}$$

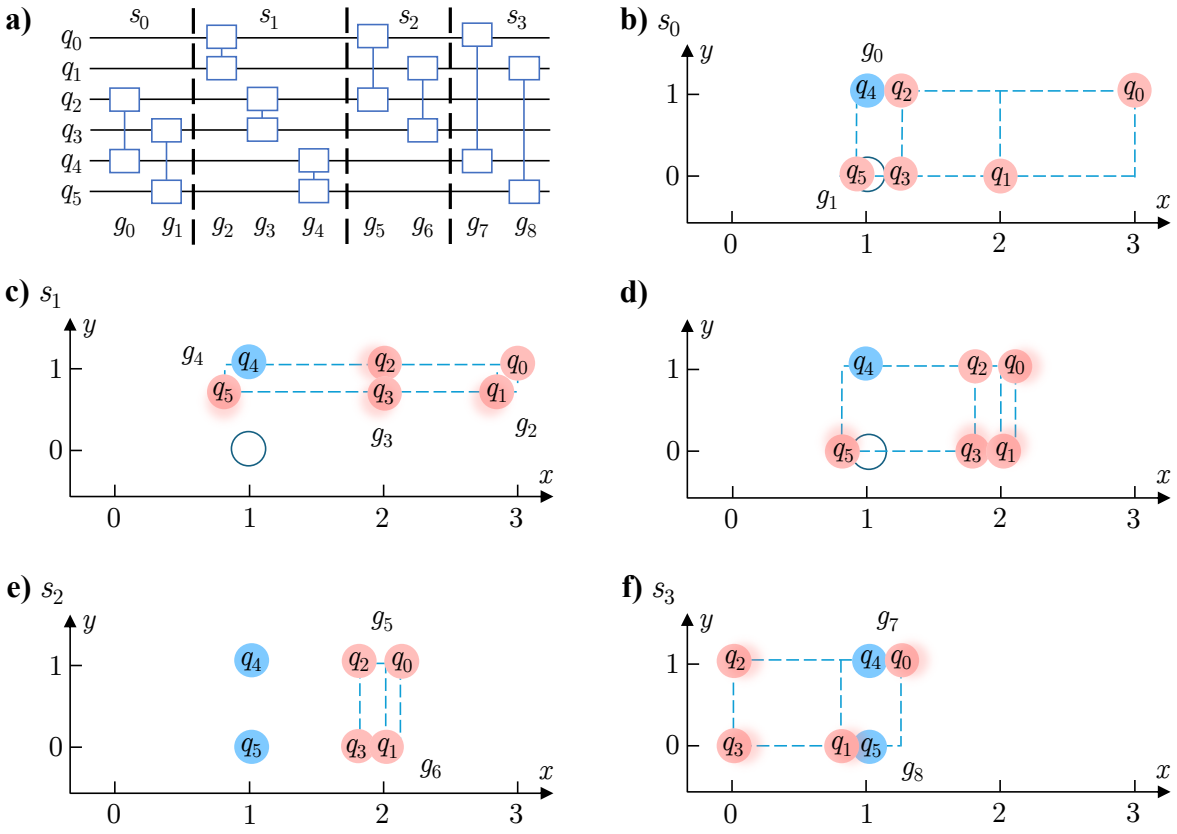


Figure 6.4: A compiled program on DPQA. **a)** The quantum circuit to compile. **b)** Stage 0. Qubits are loaded to the corresponding traps before this stage: blue qubits are in SLM, red qubits are in AOD. An AOD trap sits at every intersection of the AOD columns and rows (x and y dashed lines). An open circle represents an unoccupied SLM trap. At stage 0, (q_4, q_2) and (q_5, q_3) are at same sites to enable a Rydberg interaction. Thus, two gates g_0 and g_1 are applied to these two pairs of qubits. After stage 0, the movement shifts the lower AOD row from $y = 0$ to 1 and the middle two columns go from $x = 1$ and 2 to $x = 2$ and 3, respectively. **c)** Stage 1. Shadows of qubits indicate the direction of the movements from the previous stage to the current one. **d)** The moment after the movement between stage 1 and 2. **e)** Stage 2. q_5 is transferred from AOD to SLM (red to blue) after the movement and before stage 2 by shifting the leftmost AOD column to align with the SLM trap at $(1, 0)$ and then turning off this column. **f)** Stage 3 finishing the circuit execution.

Table 6.1: OLSQ-DPQA variable values in Figure 6.4.

gate	qubits it acts on	t_g
g_0	q_2 and q_4	0
g_1	q_3 and q_5	0
g_2	q_0 and q_1	1
g_3	q_2 and q_3	1
g_4	q_4 and q_5	1
g_5	q_0 and q_2	2
g_6	q_1 and q_3	2
g_7	q_0 and q_4	3
g_8	q_1 and q_5	3

$a_{q,s}$	q_0	q_1	q_2	q_3	q_4	q_5
s_0	1	1	1	1	0	1
s_1	1	1	1	1	0	1
s_2	1	1	1	1	0	0
s_3	1	1	0	1	0	1

$x_{q,s}$	q_0	q_1	q_2	q_3	q_4	q_5
s_0	3	2	1	1	1	1
s_1	3	3	2	2	1	1
s_2	2	2	2	2	1	1
s_3	1	1	0	0	1	1

$y_{q,s}$	q_0	q_1	q_2	q_3	q_4	q_5
s_0	1	0	1	0	1	0
s_1	1	1	1	1	1	1
s_2	1	0	1	0	1	0
s_3	1	0	1	0	1	0

$c_{q,s}$	q_0	q_1	q_2	q_3	q_4	q_5
s_0	3	2	1	1	2	0
s_1	3	2	1	1	1	0
s_2	3	2	1	1	1	1
s_3	3	2	1	1	2	3

$r_{q,s}$	q_0	q_1	q_2	q_3	q_4	q_5
s_0	1	0	1	0	1	0
s_1	1	0	1	0	0	0
s_2	1	0	1	0	0	1
s_3	1	0	1	0	1	1

For example, q_5 is in AOD at stage 0, i.e., $a_{5,0} = 1$, so when it arrives at stage 1, the row/column index remains the same, i.e., $c_{5,1} = c_{5,0}$ and $r_{5,1} = r_{5,0}$, despite the changing site indices, i.e., $y_{5,1} \neq y_{5,0}$.

Site order implying row/column order enforcing Implication 6 in the case of non-stacking rows/columns: $\forall i, i' \in [0, N),\ \forall s \in [0, S)$

$$x_{i,s} < x_{i',s} \ \Rightarrow\ c_{i,s} < c_{i',s}, \quad y_{i,s} < y_{i',s} \ \Rightarrow\ r_{i,s} < r_{i',s}. \tag{6.4}$$

For example, at stage 0, q_5 is at $x = 1$ while q_1 is at $x = 2$, so $c_{5,0} < c_{1,0}$.

No crossing between AOD row/columns enforces Implication 6 in the case of stacked rows/columns: $\forall i, i' \in [0, N),\ \forall s \in [0, S-1)$

$$\begin{aligned} (a_{i,s} == 1 \ \wedge\ a_{i',s} == 1 \ \wedge\ c_{i,s} < c_{i',s}) \ &\Rightarrow\ x_{i,s+1} \leq x_{i',s+1}, \\ (a_{i,s} == 1 \ \wedge\ a_{i',s} == 1 \ \wedge\ r_{i,s} < r_{i',s}) \ &\Rightarrow\ y_{i,s+1} \leq y_{i',s+1}. \end{aligned} \tag{6.5}$$

For example, at stage 0, q_1 is at row 0 and q_0 is at row 1, so $r_{1,0} < r_{0,0}$; at stage 1, q_1 and q_0 are both at $y = 1$, which satisfies $y_{1,1} \leq y_{0,1}$.

Maximal stacking as in Section 6.2: $\forall i, i' \in [0, N),\ \forall s \in [0, S)$

$$\begin{aligned} (a_{i,s-1} == 1 \ \wedge\ a_{i',s-1} == 1 \ \wedge\ c_{i,s-1} - c_{i',s-1} \geq C_{\text{STK}}) \ &\Rightarrow\ x_{i,s} > x_{i',s}, \\ (a_{i,s-1} == 1 \ \wedge\ a_{i',s-1} == 1 \ \wedge\ r_{i,s-1} - r_{i',s-1} \geq R_{\text{STK}}) \ &\Rightarrow\ y_{i,s} > y_{i',s}. \end{aligned} \tag{6.6}$$

(When $s = 0$, we replace the $s-1$ above with 0; otherwise, it is indeed $s-1$.) For example, at stage 0, q_5 is in column 0 while q_0 is in column 3, $c_{0,0} - c_{5,0} = 3 \geq C_{\text{STK}}$, so $x_{0,1} > x_{5,1}$, i.e., they cannot be at the same site at stage 1.

One atom, one trap. There cannot be two atoms in one trap, thus imposing Implication 1 and Implication 10. If both atoms are in AOD, either their row or column index is different; if both are in SLM, either their site x or y index is different: $\forall i \in [0, N),\ \forall i' \in [i+1, N),\ \forall s \in [0, S)$

$$\begin{aligned} (a_{i,s} == 1 \ \wedge\ a_{i',s} == 1) \ &\Rightarrow\ (c_{i,s} \neq c_{i',s} \ \vee\ r_{i,s} \neq r_{i',s}), \\ (a_{i,s} == 0 \ \wedge\ a_{i',s} == 0) \ &\Rightarrow\ (x_{i,s} \neq x_{i',s} \ \vee\ y_{i,s} \neq y_{i',s}). \end{aligned} \tag{6.7}$$

(Optional) No atom transfer by fixing array index (which is what we do in the evaluations for the optimal compiler): $\forall i \in [0, N),\ \forall s \in [0, S)$

$$a_{i,s} = a_{i,0}. \tag{6.8}$$

If it is allowed for an atom to transfer to an empty trap at the same site, i.e., forbidding transfer when there are two atoms at a site, $\forall i \in [0, N),\ \forall i' \in [i+1, N),\ \forall s \in [0, S-1)$

$$(x_{i,s+1} == x_{i',s+1} \ \wedge\ y_{i,s+1} == y_{i',s+1}) \ \Rightarrow\ (a_{i,s+1} == a_{i,s} \ \wedge\ a_{i',s+1} == a_{i',s}). \tag{6.9}$$

6.3.2.2 Circuit-Dependent Constraints

Gate collision. If two gates act on the same qubit, they cannot be executed at the same stage, e.g., g_0 and g_3 both act on q_2, so $t_0 \neq t_3$.

Gate dependence. If the order of execution between two gates cannot be changed, we ensure this by $t_j < t_{j'}$ if $g_{j'}$ depends on g_j.

Connectivity ensures Implication 8. Two qubits should be at the same site in order for an entangling gate to execute: $\forall j \in [0, G)$, g_j acting on q_i and $q_{i'}$, $\forall s \in [0, S)$

$$t_j == s \ \Rightarrow\ (x_{i,s} == x_{i',s} \ \wedge\ y_{i,s} == y_{i',s}). \tag{6.10}$$

For example, g_0 at stage 0 is on q_2 and q_4, so $x_{2,0} = x_{4,0}$ and $y_{2,0} = y_{4,0}$.

Interaction exactness enforces Implication 9. We pre-compute a list $\rho_{i,i'}$ for each pair of qubits (q_i and $q_{i'}$) that contains all the j such that g_j acts on this pair. In the example of Figure 6.4, there is only one gate g_2 acting on q_0 and q_1, so $\rho_{0,1} = \{2\}$; in contrast, there is no gates on q_0 and q_8, so $\rho_{0,8} = \emptyset$. If $\rho_{i,i'} \neq \emptyset$, then two qubits must be at the same site at some stage, and one of the gates on them is being executed at this stage: $\forall i \in [0, N),\ \forall i' \in [i+1, N),$ such that $\rho_{i,i'} \neq \emptyset,\ \forall s \in [0, S)$

$$(x_{i,s} == x_{i',s} \ \wedge\ y_{i,s} == y_{i',s}) \ \Rightarrow\ \left(\bigvee_{j \in \rho_{i,i'}} t_j == s \right). \tag{6.11}$$

Conversely, if $\rho_{i,i'} = \emptyset$, the qubits should not be at the same site ever: $\forall i \in [0, N),\ \forall i' \in [i+1, N),$ such that $\rho(i, i') = \emptyset,\ \forall s \in [0, S)$

$$x_{i,s} \neq x_{i',s} \ \vee \ y_{i,s} \neq y_{i',s}. \tag{6.12}$$

6.3.2.3 Enforcing Cardinality

There are two ways to enforce cardinality: implicitly in variable definition or explicitly with a cardinality constraint. The implicit approach is mainly for dimensions involved in the definition of the variables in the SMT model. Our arrays of variables have two dimensions: the qubit and the stage, which means whatever the model can possibly express is a computation using that many qubits and that many stages. The number of stages, S, in the optimal compiler is bounded in this approach: we only construct variables for S stages. If S is too small to execute the whole circuit, the model is unsatisfiable, so we need to add more variables. When the model becomes satisfiable, we have not introduced more variables than needed. Considering the exponential scaling of SMT solving to model size, we opt for the implicit approach to force the cardinality of stages.

An example of the explicit approach appends the SMT model with a constraint like

$$\sum_{j \in [0,G), s \in [S_{\text{LB}}, S_{\text{UB}})} \text{ITE}(t_j = s,\ 1,\ 0) \geq M, \tag{6.13}$$

where the stages between S_{LB} and S_{UB} are considered, and $\text{ITE}(\phi, w, z)$ means if the Boolean expression ϕ evaluates to true, return value w, otherwise z. Essentially, the l.h.s. is counting occurrences of a qubit pair appearing at the same site at the same stage. If this sum is larger than M, then at least M gates are executed between stages S_{LB} and S_{UB}. There are many ways to decompose the above equation to Boolean logic. We utilize the sequential counter approach offered by PySAT [IMM18] later in Figure 6.5. As a result, there are some intermediate Boolean variables introduced in the SMT model that do not correspond to any configurations of DPQA, purely for the sake of the cardinality constraint.

6.3.2.4 Scalability of the Model

The total number of variables in the optimal approach is $5NS + G$ where N is the number of qubits, S is the number of stages, and G is the number of gates. The total number of constraints is $O(G^2 + GS + N^2S)$. However, some of the variables have larger bounds. If we represent the integer variables by bit-vectors, the total number of bits to represent the variables is $NS\log(2XYRC) + G\log(S)$, where X and Y are the dimensions of the interaction site grid, C and R are the number of AOD columns and rows. The worst-case runtime of SMT solving is exponential, i.e., $O((N_{\text{SLM}}N_{\text{AOD}})^{NS} \cdot S^G)$ where $N_{\text{SLM}} = XY$ is the total number of SLM traps, and N_{AOD} is the total number of AOD traps. In the shallow circuit regime where S can be seen as a constant, and if the program is induced by sparse graphs so that $G = O(N)$, the number of bits required is $O(N\log(N_{\text{SLM}}N_{\text{AOD}}))$ and the number of constraints is $O(N^2)$. For each 'peeling' in the hybrid compiler to be introduced in Section 6.4, $S = 2$.

6.3.3 Software Implementation

With an SMT solver, we are able to not only solve valid assignments to compile circuits, but also guarantee the optimality of the solution with respect to some objective function, presented as the optimal branch in Figure 6.5. To minimize the number of Rydberg stages, S, we use relatively large spatial bounds (X, Y, R, C) which are more likely to yield satisfiable models. We start by setting S to a lower bound, e.g., the critical path in the circuit, which is 3 for the one in Figure 6.4a. If the SMT solver returns unsatisfiable, we increase S and invoke the solver again, until it finds a valid solution with S_{opt} stages. With this procedure, the optimality is guaranteed since we have checked that any smaller S yield unsatisfiable SMT models before finding the solution. If S increases beyond the number of gates, we conclude that the spatial bounds are too small and increase them. The procedure will terminate since any finite circuit of size P can be run in a finite spacetime volume bounded by $P \times P \times P$.

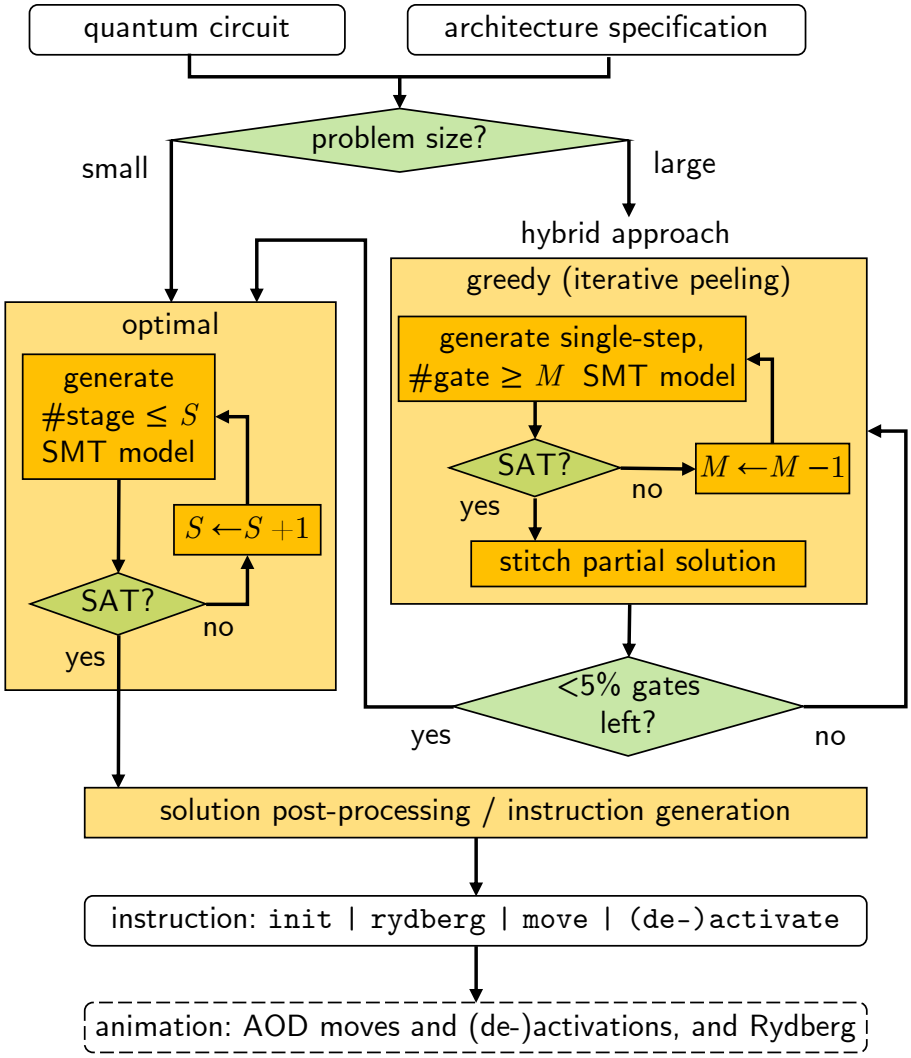


Figure 6.5: Workflow of OLSQ-DPQA. The inputs to the compiler are the quantum circuit to execute and the specifications of the DPQA architecture considered. If the problem is small, the compiler directly takes the optimal approach by constructing an SMT model where all the gates are applied to the first S stages. If the model is satisfiable, then we find a solution; otherwise, we increase S and try again. Thus, we find a solution with the minimum number of stages in the end, because lower-depth models are all checked and unsatisfiable. The SMT solution goes through a post-processing to extract the instructions for executing the quantum circuit on DPQA. *(This caption continues on the next page.)*

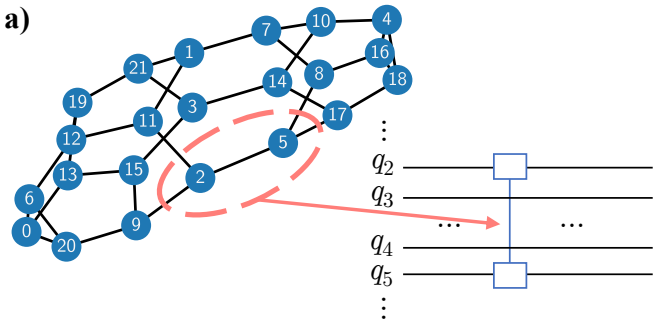
(previous page) There are only five types of instruction: `init` for initialization; `rydberg` to turn on the Rydberg laser and perform two-qubit gates; `move` for changing the coordinates of AOD rows/columns; `activate` for turning on certain AOD rows/columns for atom transfer; and `deactivate` for turning off certain AOD rows/columns. If the problem is large, the compiler takes a hybrid approach by iteratively "peeling off" the maximum number of gates possible. It generates a single-step (two-stage) SMT model with a constraint of executing more than M gates in one step. After possible decreases of M, we find the solution with as many gates executed in one step as possible. Then, we stitch this partial solution, which is one "layer peeled off", to the whole solution. When the problem becomes sufficiently small (5% of gates left), the compiler switches to the optimal approach.

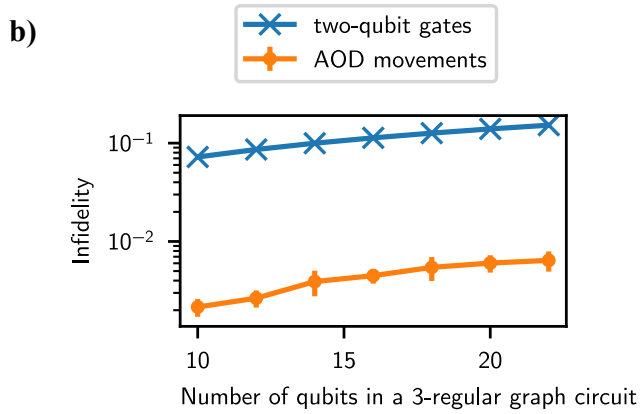
The variable assignments are not yet a DPQA executable. We need to post-process the SMT solution to produce DPQA instructions. For example, we must know the beginning and end coordinates of each AOD column, which are stored distributively in the $x_{i,s}$ and $c_{i,s}$ variables. In the example of Figure 6.4, we find q_2 is in column 1 and $x = 1$ at stage 0, i.e., $c_{2,0} = 1$ and $x_{2,0} = 1$, and $x = 2$ at stage 1, i.e., when $x_{2,1} = 2$, we infer that the AOD column 1 travels from $x = 1$ to $x = 2$. As such, the information in the SMT solution will be translated to five types of basic DPQA instructions: `init` for initial qubit loading, `rydberg` for illuminating the Rydberg laser, `move` for AOD movements, `activate` for activating AOD rows/columns, and `deactivate` for deactivating AOD rows/columns. These instructions are readily executable on DPQA, and our compiler can also generate animations from the instructions to view the execution process in action.

6.3.4 Evaluation

We benchmark the effectiveness of DPQA and our compiler, OLSQ-DPQA, on a set of quantum circuits constructed using random graphs, as illustrated in Figure 6.6a. For a given graph, we assign each node to a qubit and apply a two-qubit gate for every edge. For simplicity, we consider problems where these gates are commutable, like the controlled-R_z gates available on DPQA [LKS19, EBK23], so the compiler also explores freedom of permuting gate ordering. Compiling these circuits is more challenging compared to generic circuits due to the increased flexibility in commutation. For evaluations on realistic generic circuits, please refer to Section 6.5.

We now compare our DPQA compilation results to the compilation results on static planar architectures, where instead of physically moving qubits around, qubits are moved around using two-qubit SWAP gates. As expected, DPQA combined with the optimal compiler requires significantly fewer two-qubit gates. We tested a few compilers that perform layout synthesis for the static architecture by inserting SWAP gates: t|ket⟩ [SDC20], a heuristic compiler used in leading QAOA experiments [HSN21]; SABRE [LDX19], a heuristic compiler integrated in leading quantum programming framework, Qiskit [AAA21]; and TB-OLSQ2 [LKT23], a leading near-optimal compiler for static architectures. The gaps of the two-qubit gate count for QAOA benchmarks with 22 nodes in the graph are 4.5x, 2.5x, and 1.7x, respectively.


a)
q_2
q_3
q_4
q_5


b)
two-qubit gates
AOD movements
Infidelity
10^{-1}
10^{-2}
10
15
20
Number of qubits in a 3-regular graph circuit

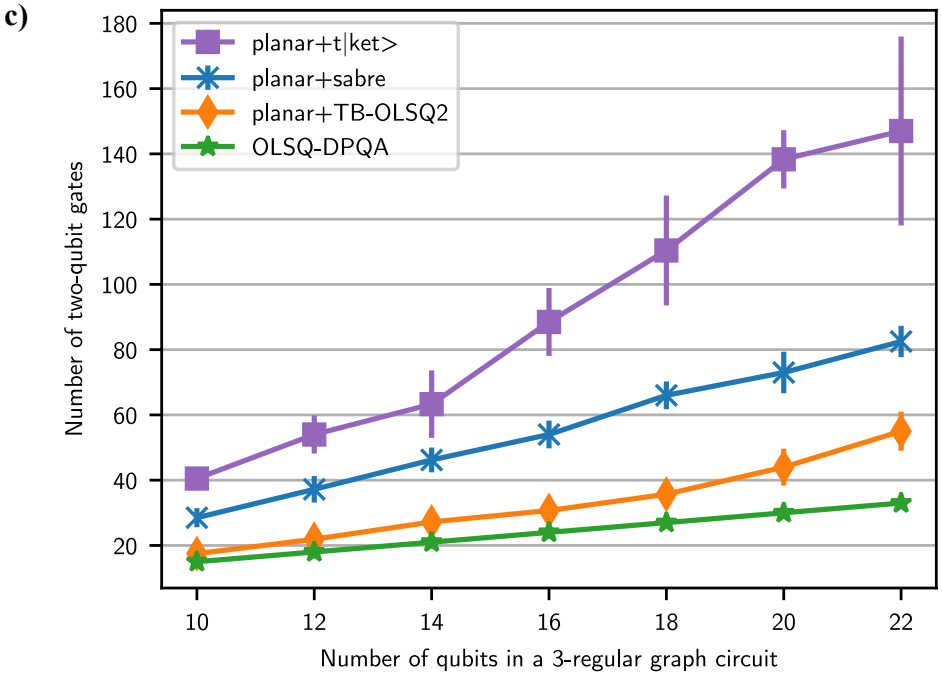

c)
planar+t|ket>
planar+sabre
planar+TB-OLSQ2
OLSQ-DPQA
Number of two-qubit gates
180
160
140
120
100
80
60
40
20
10
12
14
16
18
20
22
Number of qubits in a 3-regular graph circuit

Figure 6.6: *(previous page)* Evaluation of the optimal layout synthesis in OLSQ-DPQA. **a)** Graph circuits. Given any graph, we treat each node as a qubit and add a two-qubit entangling gate for every edge in the graph to construct the graph circuit. We assume the gates are commutable, so gate order does not matter. The benchmarks used are graph circuits generated by 3-regular graphs of size 10 to 22. For each size, we have 10 random graphs. **b)** Comparison of infidelity caused by the Rydberg laser (performing two-qubit gates) and the AOD movements. The latter is 27x smaller on average. We make such an estimation using 99.5% two-qubit gate fidelity [EBK23] and a movement scheme that yields low atom heating as in [BLS22]. **c)** Comparison of the number of two-qubit gates required on a static planar architecture (Google's Sycamore) and DPQA employing different compilers. Error bars are standard deviations among 10 random graphs of the same size. The compilers are t|ket⟩ [SDC20], SABRE [LDX19] (integrated in Qiskit [AAA21]), and TB-OLSQ2 [LKT23]. TB-OLSQ2 is near-optimal for static architectures, but there is still a significant gap (1.7x) between it and the optimal DPQA compiler, which mainly comes from SWAP gates inserted on the static architecture, each requires three entangling gates (controlled R_z) [GJE20].

6.4 Hybrid Approach

The runtime for SMT solving scales exponentially in the worst case, so the optimal compiler can take a very long time to solve certain cases, as seen in Figure 6.8b. Due to the complicated constraints, it is also challenging to design near-optimal purely heuristic algorithms to search the solution space of DPQA. Therefore, we adopt a two-level approach, as illustrated in the hybrid approach in Figure 6.5. For large problems, at the higher level, we apply a greedy heuristic in that, at every stage, we find the movement to maximize the number of gates to execute in the next stage. We repeat this process until there are a sufficiently small number of gates remaining and then switch to the optimal approach. This technique is inspired by 'iterative peeling' for multi-layer routing of classical circuits [CHS93].

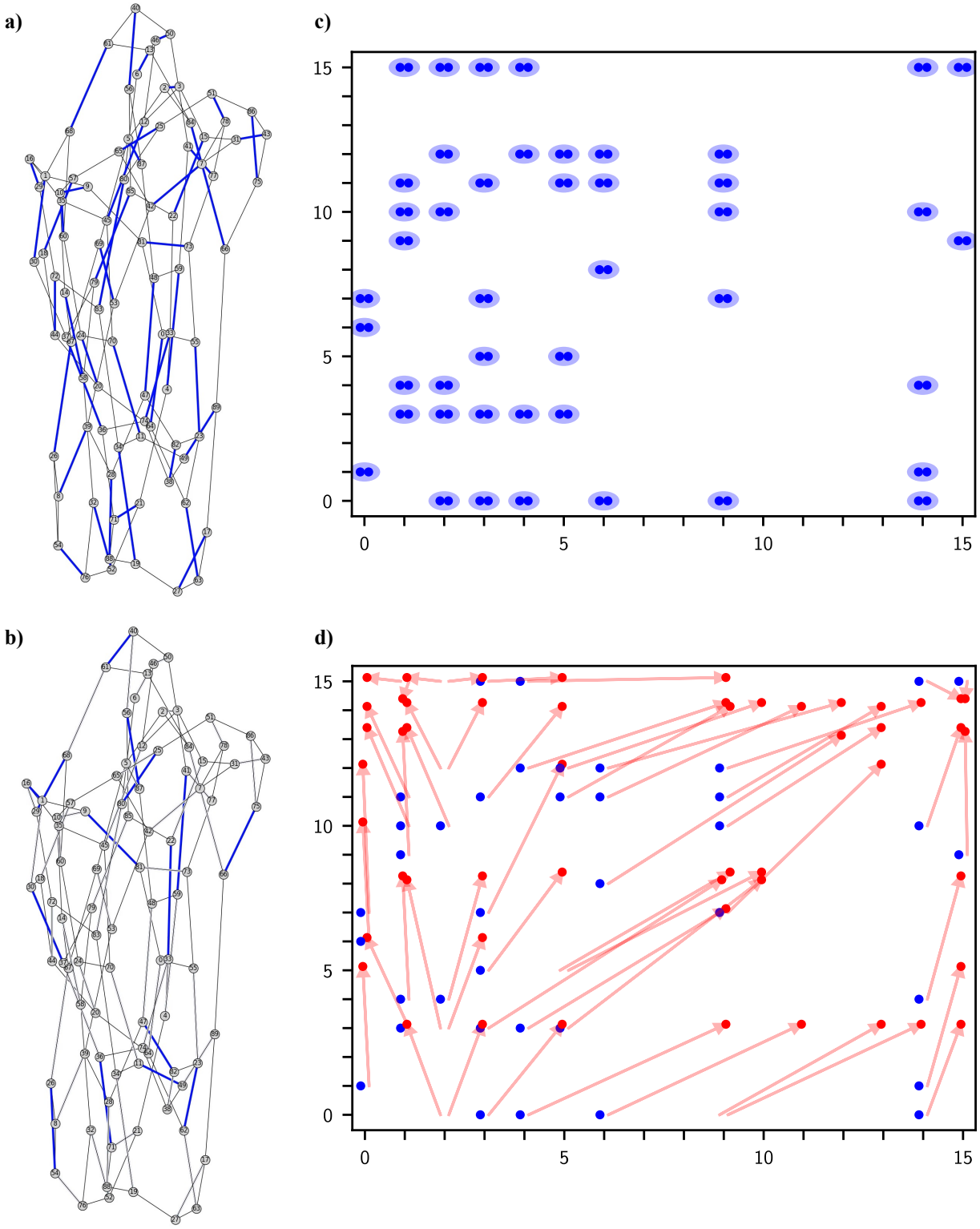
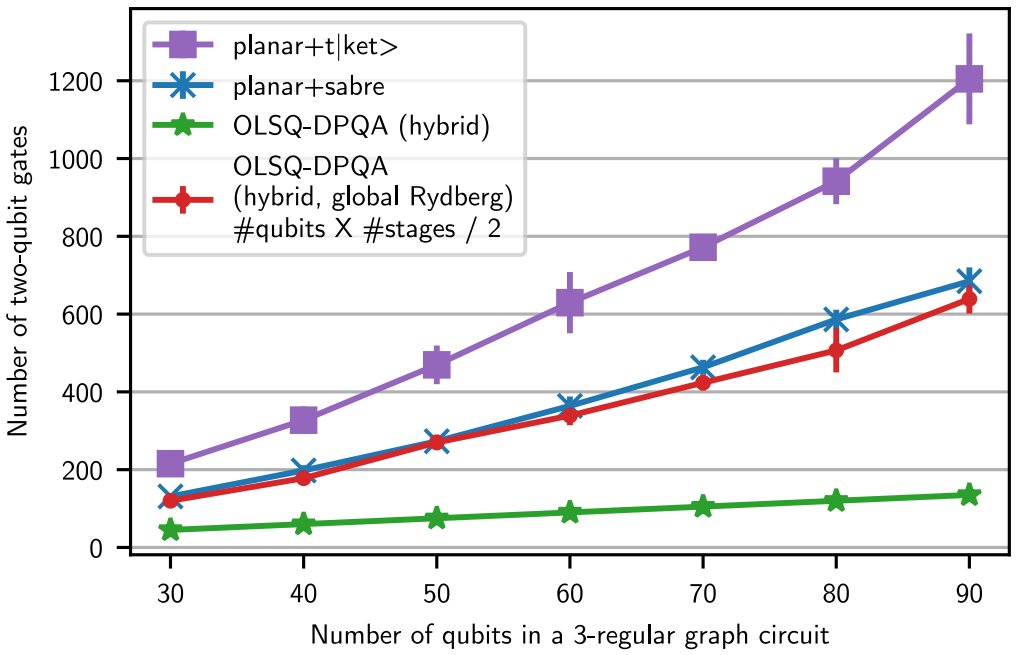

a)
c)
b)
d)
15
10
5
0
0
5
10
15

Figure 6.7: *(previous page)* One of the largest benchmarks we are able to compile with OLSQ-DPQA. **a,b)** The 90-node 3-regular graph. The highlighted edges are gates executed at the stages in c) and d), respectively. **c)** One stage of the compiled result. The dots are qubits in SLM. The ovals indicate two-qubit gates performed at this stage, which have a 1-to-1 correspondence with the edges in a). After this stage, some qubits are transferred to AOD and moved. **d)** The next stage. The red dots are the AOD qubits, and the arrows indicate the parallel movements from c) to the current state. Readers are welcome to check out our code base for this animation.

Specifically, if there are still gates to execute, we construct a "single-step" SMT model with two stages and set the qubit location of the initial stage to that of the current stage in the full solution. For instance, suppose the compiler has already processed g_0 to g_4 in Figure 6.4a and progressed to stage 1 (Figure 6.4c). In the single-step model, all initial locations are set by the previous partial solution, e.g., $x_{4,0} = 1$ since q_4 at $x = 1$ in Figure 6.4c. Then, we optimize the number of gates executed in the second stage in the single-step model with a procedure similar to the one in the optimal approach, except that we decrease the number of gates to execute in the second stage from an upper bound of M instead of increasing the number of stages from a lower bound. The upper bound is the size of the maximum matching of the graph constructed from the remaining gates. In our example, the remaining gates g_5 and g_7 both act on q_0, whereas g_6 and g_8 both act on q_1, so only one gate in each of these two pairs can be executed together, i.e., the size of the maximum matching is 2. The compiler appends the single-step model with a constraint that says there are at least 2 gates executed at the next stage and invokes the SMT solver, which can find such a solution (Figure 6.4e). We stitch this partial solution to the full solution, remove gates g_5 and g_6 which is a layer of gates "peeled off", and continue to the next "peeling". If there are only a few gates remaining (we opt for 5%) the compiler switches to the optimal approach to solve for the final stages.

a)


planar+t|ket>
planar+sabre
OLSQ-DPQA (hybrid)
OLSQ-DPQA
(hybrid, global Rydberg)
#qubits X #stages / 2
Number of two-qubit gates
0
200
400
600
800
1000
1200
30
40
50
60
70
80
90
Number of qubits in a 3-regular graph circuit

b)

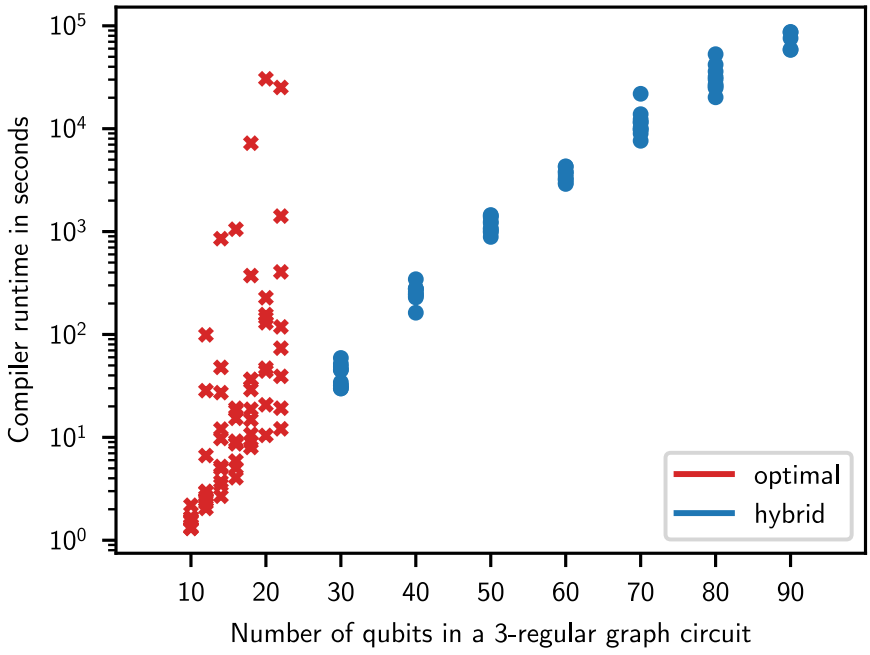

Compiler runtime in seconds
10^0
10^1
10^2
10^3
10^4
10^5
optimal
hybrid
10
20
30
40
50
60
70
80
90
Number of qubits in a 3-regular graph circuit

Figure 6.8: *(previous page)* Evaluation of the greedy-optimal hybrid approach in OLSQ-DPQA. **a)** Comparison of the number of two-qubit gates required on a static planar architecture (10x10 grid) using different compilers and DPQA. For DPQA, the number of two-qubit gates scales as n, whereas for the state-of-the-art heuristic solver on the static planar architecture, SABRE, scales as $n^{1.52\pm0.02}$ where n is the number of qubits. DPQA requires far fewer two-qubit gates, 5.1x less than SABRE, and scales linearly. **b)** Comparison of runtime of the optimal and hybrid approaches in OLSQ-DPQA. The benchmarks are graph circuits with 10, 12, 14, 16, 18, 20, 22, 30, 40, 50, 60, 70, 80, and 90 qubits. We generated 10 3-regular graphs of each size. In OLSQ-DPQA, we set the spatial bounds to $X = Y = R = C = 16$. We used a desktop computer with an Intel Core i7-10700KF CPU and 32 GB RAM and set the time limit to 10^5 seconds which is approximately a day. Note that the compiler runtime can vary depending on the specific hardware and environment where it is run. The timeout instances are 20_5, 22_5, and 22_8 for the optimal approach, 80_1, 90_0, 90_2, 90_6, 90_8, and 90_9 for the hybrid approach, where the subscripts are the indices of the graph. All the random graphs used are provided in the code base. Since both of them internally rely on SMT solving, the worst-case runtime scalings are both exponential in the size of the graph with which we generate the quantum circuit. However, the hybrid approach is significantly faster so that large instances can be solved (up to 90 qubits in time limit). Compared to the optimal approach, the scaling of the hybrid approach is mainly related to size rather than the specific graph, which is demonstrated by the much smaller spread of data points at each size.

The hybrid approach cannot fundamentally improve the runtime scaling from exponential to polynomial because it still relies on SMT solving, but it greatly accelerates the process, with some sacrifice on optimality. As exhibited in Figure 6.8b, it is much faster than the optimal compiler and the divergence of runtime within benchmarks of the same size is also much smaller. Within a reasonable amount of time (10^5 s $\approx$ a day), the hybrid compiler managed to compile some 90-qubit circuits whereas the optimal compiler, in the worst case, can only compile up to the 22-qubit circuit. We present one of the largest circuits compiled in Figure 6.7 where part a and b exhibit the graph generating the quantum circuit which has a complex connectivity, while part c and d are two stages in the program execution.

This hybrid approach is implemented in the OLSQ-DPQA, which is open-source under the BSD 3-clause license.[1] The code base includes Python scripts that 1) generate the SMT models and iteratively invoke an SMT solver, Z3 [dB08] to solve them, 2) generate DPQA instructions and animations based on SMT solutions, 3) draw plots in the evaluations. The dependencies are Python packages `z3-solver 4.12.1.0` [dB08], `python-sat 0.1.8.dev1` [IMM18], `networkx 3.0` [HSS08], and `matplotlib 3.6.2` [Hun07]. The code base also includes all SMT solutions in the evaluations and some example animations.

We compare the required number of two-qubit gates by DPQA and a static planar architecture (10x10 grid) in Figure 6.8a. We find that the savings from DPQA on such a large system is significant compared to the static architecture: 5.1x and 8.9x reduction in the number of two-qubit gates, respectively, compared to the compilation results by SABRE [LDX19] (as in `qiskit 0.42.1` [AAA21]) and t|ket⟩ [SDC20] (as in `pytket 1.13.2`). If the heuristics place qubits in an $\sqrt{n}$-by-$\sqrt{n}$ region, each gate may require $O(\sqrt{n})$ SWAPs to route. Then, for $O(n)$ gates, as in our benchmark set, $O(n^{1.5})$ SWAPs are required. We observe this scaling in the results of SABRE: with a log-log fitting, the number of two-qubit gates scales in the 1.52 ± 0.02 power of the number of qubits. In comparison, DPQA routes the gates by AOD movements instead of SWAPs, so the number of gates scales linearly.

[1] https://github.com/UCLA-VAST/DPQA

Comparing the number of two-qubit gates is most suitable if the DPQA is equipped with an individually addressable Rydberg laser (or other methods of turning off the Rydberg excitation locally) that does not accumulate the same error on idling qubits (e.g., in [GSS22]). Instead, if DPQA is equipped only with a global Rydberg laser that illuminates the whole plane, although the number of two-qubit gates is greatly reduced, the effective number of two-qubit gates, i.e., the number of qubits times the number of stages divided by 2, is only slightly better (7%) than the SABRE results on the static architecture *assuming that* a global Rydberg laser induces the similar error rate, at every stage, on idling qubits as well as qubits involved in two-qubit gates [BLS22].

6.5 Handling Generic Quantum Circuits

Previously, our attention was primarily on the compilation of circuits comprised of commutable two-qubit gates. We find that these circuits showcase the massive parallelism of DPQA architecture. Also, the flexibility in commutation adds extra challenges to the compilation problems. In *generic circuits*, e.g., Figure 6.9a, there are two notable differences. Firstly, these circuits include single-qubit gates (e.g., g_0 and g_1). Secondly, the gates in these generic circuits are not necessarily commutable. We assume a *dependency* in cases where two gates act on the same qubit, dictating a fixed order; for instance, g_0 and g_3 both acting on q_1 means g_3 must be scheduled after g_0. Our software implementation includes an `all_commutable` flag as part of the problem specification. When this flag is inactive, OLSQ-DPQA defaults to the workflow illustrated in Figure 6.9c: prior to compilation, we remove all single-qubit gates to derive the dependency graph of two-qubit gates, as shown in Figure 6.9b. Due to the dependencies, only the front layer of the graph, represented by the red nodes (e.g., g_2 and g_3 initially), can be processed. OLSQ-DPQA compiles the qubit movements for these gates, maximizing the number of executed gates, and removes them from the dependency graph (grayed out nodes). Sometimes, not the entire front layer is

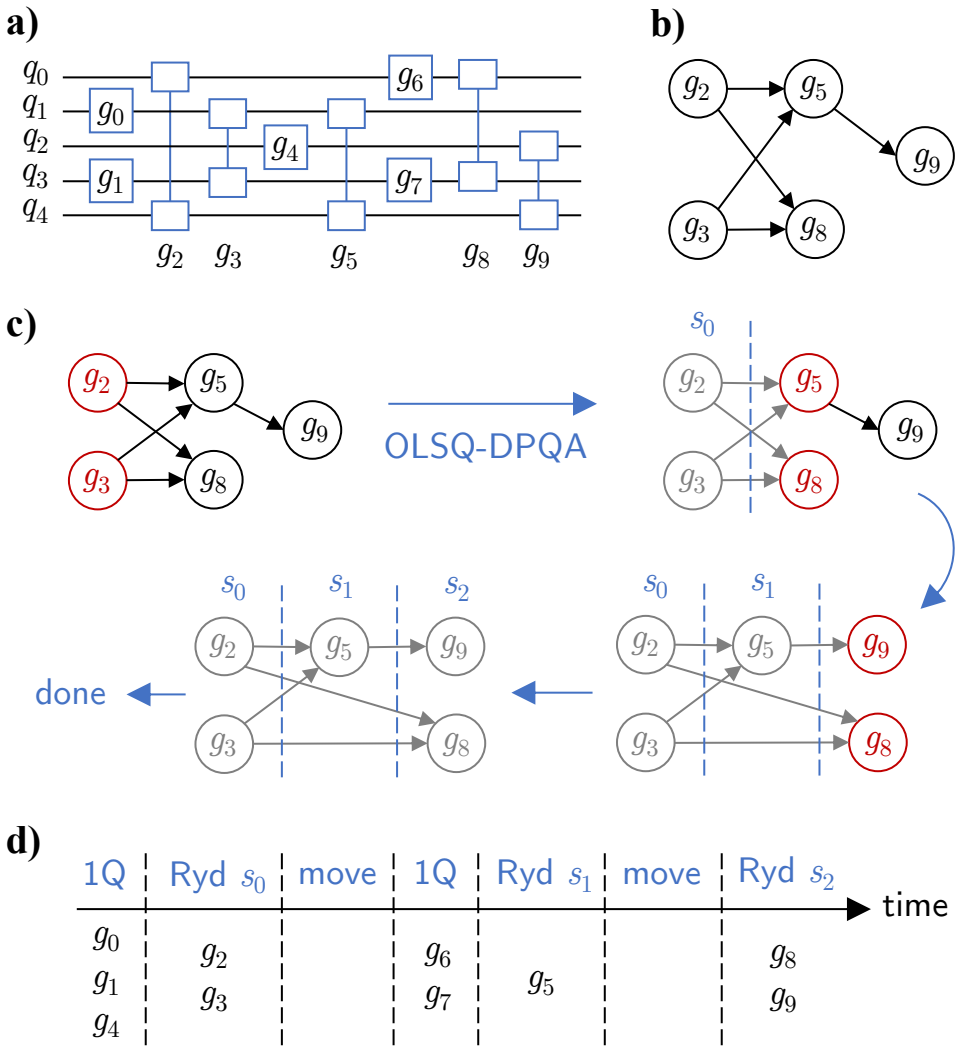


Figure 6.9: Handling generic circuits in OLSQ-DPQA. **a)** Example circuit. **b)** Dependency graph of two-qubit gates. **c)** Compilation process. OLSQ-DPQA is invoked 3 times. Each time, only the front layer (red nodes) is processed. It is possible the entire front layer is not executed, leading to the inclusion of the remaining nodes in the subsequent front layer (e.g., g_8). **d)** Final result. Prior to each Rydberg stage, we execute all single-qubit gates that have no dependency to any gates not yet executed.

executed depending on the qubit locations (e.g., g_5 is executed at s_1 while g_8 is not), leaving the remaining gates for the next round. This process continues until all nodes are processed. Finally, we reintroduce the single-qubit gates, as depicted in Figure 6.9d. Prior to each two-qubit gate stage, we execute all single-qubit gates without dependencies at this point. For instance, g_7 only depends on g_3, which is executed at s_0, allowing g_7 to be executed before s_1.

We benchmark OLSQ-DPQA on realistic generic circuits from QASMBench [LSK22], detailed in Table 6.2. Specifically, we picked all the 'medium' and 'large' benchmarks with fewer than 100 qubits and less than 1000 gates. Certain benchmarks share the same circuit family but differ in size, such as various-sized adders. The *2Q depth* of a circuit is the length of the longest path in the two-qubit dependency graph like Figure 6.9b. For a static 10x10 grid qubit coupling graph, we utilized SABRE [LDX19] within Qiskit [AAA21] to layout qubits and insert SWAPs. In contrast, OLSQ-DPQA relies solely on qubit movement to route qubits, resulting in a reduction of two-qubit gates by 1.8x geomean, as shown in the rightmost column of Table 6.2. While, in most instances, the number of two-qubit stages (Rydberg) aligns closely with the 2Q depth of the circuit, OLSQ-DPQA may require a larger number of stages. This arises from the fact that not all gates in the front layer can be executed at every stage due to the specific qubit locations at that point. These front layers are generally less complicated than random graphs. Consequently, even in cases where these benchmarks have more gates than graph circuits in previous sections, the compiler runtime tends to be shorter.

Table 6.2: OLSQ-DPQA results on QASMBench [LSK22]. We pick all of their 'medium' and 'large' benchmarks with less than 100 qubits and less than 1000 gates. '2Q depth' of a circuit is the length of the longest path in the two-qubit dependency graph. The number of Rydberg stages in the OLSQ-DPQA results is close to 2Q depth (1.13x geomean). The SABRE results assume a 10x10 grid qubit coupling graph. OLSQ-DPQA reduces two-qubit gates because it uses movements instead of SWAPs to route qubits. These experiments were run on a server with two AMD EPYC 7V13 CPUs and 512 GB RAM.

	Benchmark statistics			SABRE	OLSQ-DPQA			**Reduction**
Name	**Qubits**	**2Q gates**	**2Q depth**	**2Q gates**	**2Q gates**	**Rydbergs**	**Runtime/s**	**2Q gates**
seca	11	84	41	129	84	41	2.07E+1	1.54x
sat	11	252	204	444	252	205	1.05E+2	1.76x
cc	12	12	12	21	12	12	7.41E+0	1.75x
multiply	13	40	23	64	40	23	1.63E+1	1.60x
gcm	13	762	762	1257	762	762	5.27E+2	1.65x
bv	14	13	13	25	13	13	1.10E+1	1.92x
qf21	15	115	112	202	115	112	1.07E+2	1.76x
multiplier	15	222	133	414	222	137	1.33E+2	1.86x
dnn	16	384	48	384	384	53	6.47E+1	1.00x
qec9xz	17	32	12	62	32	16	2.03E+1	1.94x
qft	18	306	66	549	306	82	1.20E+2	1.79x
bigadder	18	130	88	220	130	89	1.32E+2	1.69x
square_root	18	898	644	1909	898	651	8.52E+2	2.13x
bv	19	18	18	39	18	18	2.88E+1	2.17x
qram	20	136	80	247	136	82	1.49E+2	1.82x
cat_state	22	21	21	39	21	21	4.76E+1	1.86x
ghz_state	23	22	22	40	22	22	5.55E+1	1.82x
swap_test	25	96	63	147	96	63	1.90E+2	1.53x

(This table continues on the next page)

(previous page)

	Benchmark statistics			SABRE	OLSQ-DPQA			**Reduction**
Name	**Qubits**	**2Q gates**	**2Q depth**	**2Q gates**	**2Q gates**	**Rydbergs**	**Runtime/s**	**2Q gates**
knn	25	96	63	144	96	63	1.89E+2	1.50x
ising	26	50	4	59	50	7	2.33E+1	1.18x
wstate	27	52	28	67	52	28	1.02E+2	1.29x
adder	28	195	97	321	195	98	3.90E+2	1.65x
adder	64	455	181	845	455	188	8.32E+3	1.86x
bv	30	18	18	42	18	18	8.66E+1	2.33x
bv	70	36	36	108	36	36	2.72E+3	3.00x
cat	35	34	34	58	34	34	2.30E+2	1.71x
cat	65	64	64	154	64	64	3.05E+3	2.41x
cc	32	32	32	95	32	32	1.77E+2	2.97x
cc	64	64	64	202	64	64	2.96E+3	3.16x
dnn	33	248	95	365	248	97	5.59E+2	1.47x
dnn	51	392	140	632	392	152	2.91E+3	1.61x
ghz	40	39	39	87	39	39	3.86E+2	2.23x
ghz	78	77	77	215	77	77	1.02E+4	2.79x
ising	34	66	4	84	66	10	7.65E+1	1.27x
ising	66	130	4	205	130	12	1.46E+3	1.58x
ising	98	194	4	347	194	17	3.25E+4	1.79x
knn	31	120	78	186	120	78	4.07E+2	1.55x
knn	67	264	168	486	264	168	8.14E+3	1.84x
qft	29	812	110	1547	812	178	7.37E+2	1.91x
qugan	39	296	102	467	296	113	1.07E+3	1.58x
qugan	71	552	182	936	552	193	1.36E+4	1.70x
swap_test	41	160	103	277	160	103	1.15E+3	1.73x
swap_test	83	328	208	628	328	208	3.71E+4	1.91x
wstate	36	70	37	94	70	37	3.01E+2	1.34x
wstate	76	150	77	273	150	77	9.40E+3	1.82x
geomean								1.8x

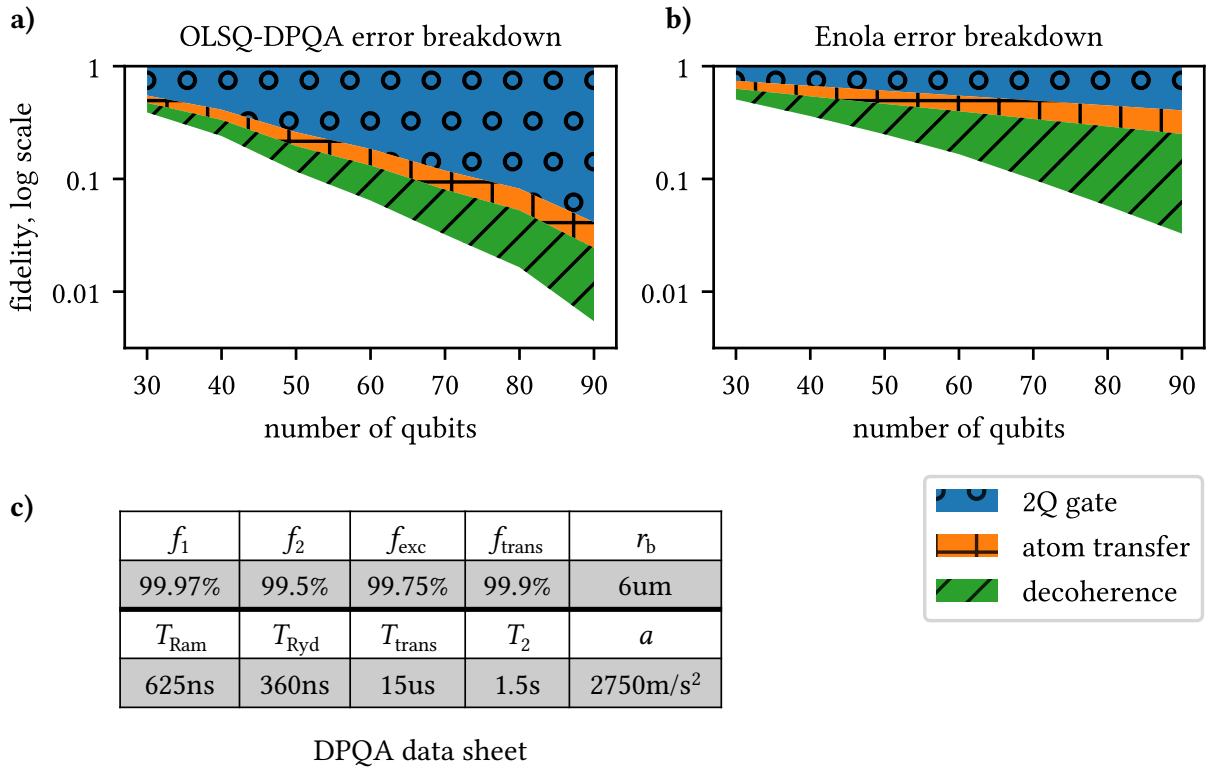
CHAPTER 7

Enola: Efficient and Near-Optimal Layout Synthesis for Dynamically Field-Programmable Qubit Arrays

The neutral atom hardware is advancing in a fast pace, as discussed in Section 1.2. Although OLSQ-DPQA provides a solid foundation to the layout synthesis for DPQA, it does not scale to the current maximum hardware capability. Additionally, as mentioned in Section 2.2.2, there are some known compromises in the design of OLSQ-DPQA, which result in fidelity decrease as we shall analyze in Section 7.1. This leads us to rethink the formulation and decouple it into three tasks: scheduling in Section 7.3, placement in Section 7.4, and routing in Section 7.5. Specifically, the scheduling is based on a provably near-optimal graph edge coloring algorithm, Misra-Gries, which is introduced in Section 7.2. Based on the refined formulation, we develop a new layout synthesis tool for DPQA, Enola, featuring efficient and high-quality solution, as we will see in the evaluations in Section 7.6.

7.1 Motivation: Fidelity Analysis

We model three error sources: imperfect gates, atom transfers, and qubit decoherence. The parameters follow leading experiments [BLS22, BLS24] and are summarized in Figure 7.1c. Single-qubit gates have the fidelity $f_1 = 99.97\%$ and the duration $T_{\text{Ram}} = 625$ ns. These gates can be individually addressed to corresponding qubits [TPC24], so there are no side effect errors on other qubits. We make the same assumption as in Section 6.5 that the single-qubit gates are first removed so that *the compiler only handles the two-qubit gates*. Then,



f_1	f_2	f_{exc}	f_{trans}	r_b
99.97%	99.5%	99.75%	99.9%	6um
T_{Ram}	T_{Ryd}	T_{trans}	T_2	a
625ns	360ns	15us	1.5s	2750m/s^2

Figure 7.1: OLSQ-DPQA and Enola error breakdown. The benchmarks are 3-regular Max-Cut QAOA circuits used in [TBL24]. For the 90-qubit circuits, Enola reduces two-qubit gate stages by 3.7x and improves the overall fidelity by 5.9x.

the single-qubit gates are inserted back to the compiled results. Two-qubit gates have the fidelity $f_2 = 99.5\%$ and the duration $T_{\text{Ryd}} = 360$ ns. The other qubits are also excited by the Rydberg laser, e.g., q_5 in Figure 2.4b, each has the fidelity $f_{\text{exc}} = 99.75\%$. Atom transfers have the fidelity $f_{\text{trans}} = 99.9\%$ and the duration $T_{\text{trans}} = 15$ us. Note that multiple transfers can be simultaneous, e.g., the three transfers in Figure 2.4a are simultaneous and take 15 us. The coherence time of qubits is $T_2 = 1.5$ s. The decoherence effect of a qubit q is modelled by a multiplicative factor $1 - T_q/T_2$ where T_q is its idling time, i.e., the total duration of the procedure carried out on DPQA minus any time spent on gates or transfers. The majority of T_q is spent on AOD movements. The move distance, d, and time, t, follow the relation $d/t^2 = a = 2750$ m/s^2 [BLS22], e.g., if $d = 110$ um, then $t = 200$ us.

The overall fidelity is computed by

$$f = (f_1)^{g_1} \cdot \overbrace{(f_2)^{g_2} \cdot (f_{\text{exc}})^{|Q|S-2g_2}}^{\text{two-qubit gate}} \cdot \overbrace{(f_{\text{trans}})^{N_{\text{trans}}}}^{\text{atom transfer}} \cdot \overbrace{\prod_{q \in Q} (1 - T_q/T_2)}^{\text{decoherence}}, \tag{7.1}$$

where g_1 and g_2 are the number of single-qubit and two-qubit gates, respectively, Q is the set of qubits, S is the number of stages, $|Q|S - 2g_2$ calculates the qubits affected by the Rydberg laser but does not perform a gate, and N_{trans} is the total number of atom transfers. Since we only focus on two-qubit gates, the term $(f_1)^{g_1}$ is a constant and we ignore it from now on. As an example, we calculate the fidelity for the process in Figure 2.4. There are 3 two-qubit gates so $(f_2)^{g_2} = 0.9950^3 = 0.9851$. Only q_5 is excited but does not perform a gate so $(f_{\text{exc}})^{|Q|S-2g_2} = f_{\text{exc}}^{7\times1-2\times3} = 0.9975$. Thus, the total *two-qubit gate term* is $0.9851 \times 0.9975 = 0.9826$. Since there are 3 atom transfers in Figure 2.4a, the *atom transfer term* is $(f_{\text{trans}})^{N_{\text{trans}}} = 0.9990^3 = 0.9970$. The longest movement belongs to q_4: it travels a $\sqrt{2}$ site separation, i.e., $\sqrt{2} \times 2.5r_b = 21.21$ um. Thus, the AOD movement from Figure 2.4a to Figure 2.4b takes $t = (21.21 \text{ um} \ / \ 2750 \text{ m/s}^2)^{0.5} = 87.82$ us. This is the T_q for the moving qubits q_0, q_4, and q_5. The other four qubits are additionally idling during the atom transfer, so their $T_q = 87.82 \text{ us} + T_{\text{trans}} = 102.82$ us. Therefore, the *decoherence term* is $[1 - 87.82/(1.5 \times 10^6)]^3 \times [1 - 102.82/(1.5 \times 10^6)]^4 = 0.9996$. Finally, the overall fidelity is $f = 0.9826 \times 0.9970 \times 0.9996 = 97.92\%$.

In Section 6.4, OLSQ-DPQA compiles a set of QAOA circuits designed for the MaxCut problem on 3-regular graphs with the number of qubits ranging from 30 to 90. We evaluate the compiled results with our fidelity model and present the breakdown in Figure 7.1a. Note that, to draw the figure, we take the logarithm of the fidelity terms so that they are additive. At 90 qubits, the two-qubit gate fidelity term is 0.0414, the atom transfer term is 0.592, and the decoherence term is 0.223. Thus, the dominating error source are the two-qubit gates. However, there is a gap between the number of stages achieved by OLSQ-DPQA, on average 14.6 for 90 qubits, and the theoretical lower bound, 3, because each qubit is only involved in 3 two-qubit gates. With our compiler, Enola, only 4 stages are produced, pushing the

two-qubit fidelity term to 0.406. This effect is evident in the great decrease of the two-qubit gate portion in Figure 7.1b compared to Figure 7.1a.

7.2 Misra-Gries Algorithm for Edge Coloring

For a graph $G = (V, E)$, an *edge coloring* is a function $\phi : E \to \mathbb{Z}$ that assigns different values to edges incident on the same node, ensuring $\phi(e) \neq \phi(e')$ for $e, e' \in E$ where $e \neq e'$ and $e \cap e' \neq \emptyset$. We will use the coloring problem shown in Figure 7.2a as the running example in this section. The minimum number of colors required to achieve an edge coloring is termed the *chromatic index*, $\chi'(G)$. Vizing's theorem [Viz65] states that, for any simple undirected graph G, $\Delta(G) \leq \chi'(G) \leq \Delta(G) + 1$, where $\Delta(G)$ is the maximum degree of any node in G. The algorithm by Misra and Gries [MG92] is a constructive proof of Vizing's theorem that computes an edge coloring in $O(|V| \cdot |E|)$ time, using no more than $\Delta(G) + 1$ colors, at most one more than optimal. In our example, the algorithm produces a coloring with four colors, as shown in Figure 7.2o, while an optimal solution in Figure 7.2b uses three colors.

Before delving into the algorithm, we define some key concepts. A color is *free* on a node u if no incident edge of u uses this color. A *fan* $F[1 : k]$ for a node X is a non-empty sequence of distinct nodes where $(X, F[k])$ is uncolored and the color of each edge $(X, F[i-1])$ is free at node $F[i]$ for $1 < i \leq k$. An example fan for node 3 includes nodes 2 and 5 (Figure 7.2c). Throughout Figure 7.2, purple dots indicate nodes in a fan, and purple arrows indicate the order within a fan. *Rotating a fan* involves shifting the color of edge $(X, F[i-1])$ to $(X, F[i])$ for $1 < i \leq k$, and leaving $(X, F[1])$ uncolored. This operation maintains a valid coloring:

1. Removing the color from $(X, F[1])$ is valid since it becomes uncolored.
2. Since the color of $(X, F[i-1])$ is free at node $F[i]$, it can be assigned to $(X, F[i])$.
3. Node X simply experiences a rotation of colors without the introduction of new colors.

A *maximal* fan of X is one that cannot be extended by adding more neighbors of X. For

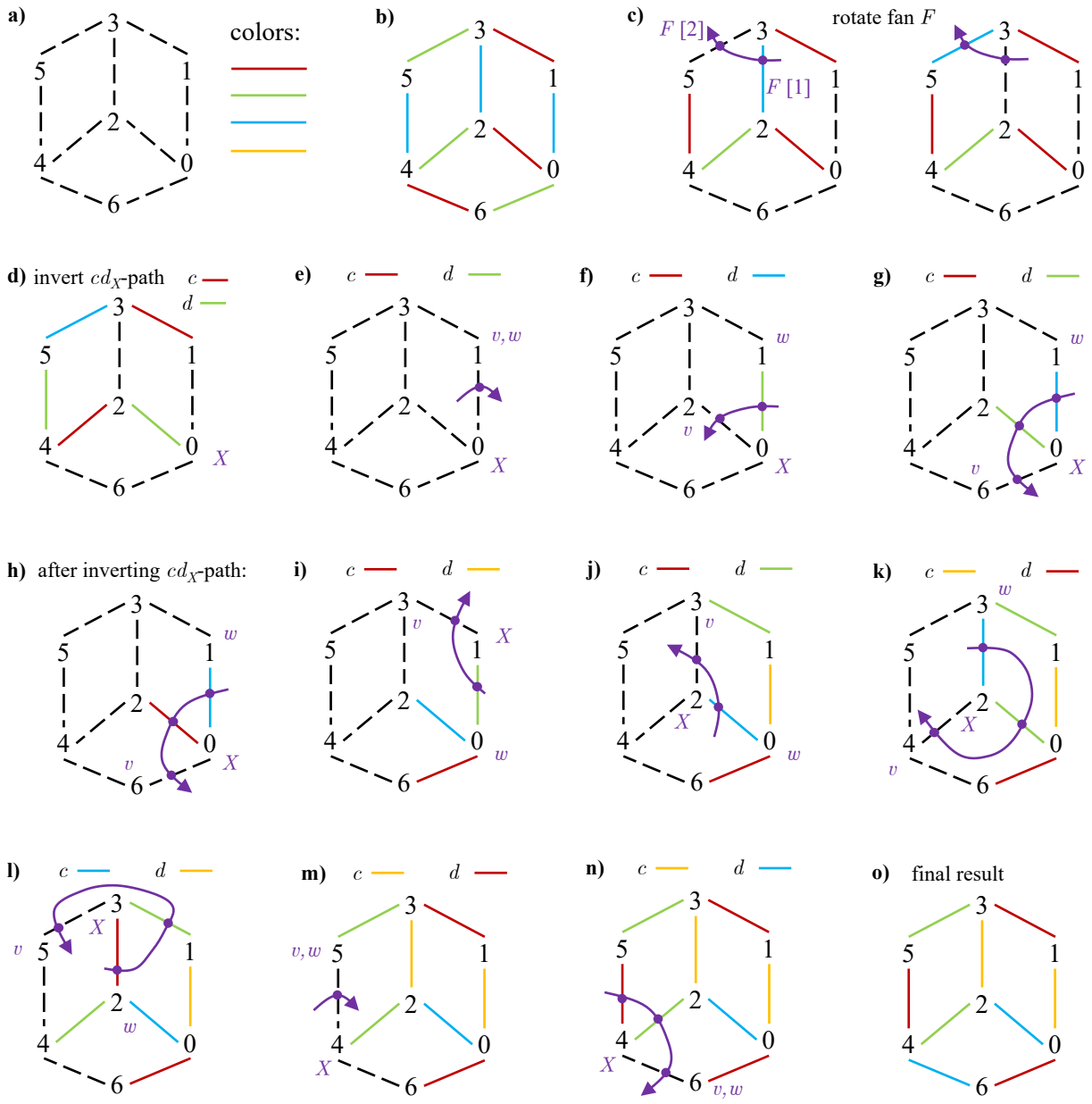


Figure 7.2: Misra-Gries algorithm. **a)** Edge coloring problem. Dashes means uncolored. **b)** The optimal solution with three colors. **c)** Rotating a fan F. **d)** Inverting a cd_X-path where c is red, d is green, and X is 0. **e-o)** Step-by-step execution of the Misra-Gries Algorithm. The final result has four colors. This algorithm guarantees one more color from optimal.

instance, in Figure 7.2c, the fan at node 3 is maximal because the color red from edge $(3, 1)$ is not free on nodes 2 or 5, preventing the inclusion of node 1. A *subfan* of $F[1 : k]$ includes a contiguous sequence of nodes from the fan, starting from any $F[w]$, $w \in \{1, ..., k\}$, to $F[k]$.

A cd_X-path is a maximal path through node X, alternating between colors c and d. Inverting this path involves swapping c and d along the path, resulting in a valid coloring:

1. For nodes on the path but not endpoints, swapping colors of two incident edges maintains valid coloring.

2. If a node u is an endpoint, and suppose edge (u, v) with color c is on the path, then d must be free for u. Otherwise, there is some edge (u, w) with color d which can be added to the path, violating the maximality in the definition of a cd_X-path. Since d is free for u, we can switch the color of (u, v) to d.

The benefit of inverting a cd_X-path is that if X is an endpoint, inverting the path frees a specific color for X. For example, Figure 7.2d exhibits the coloring after switching red-green path 0-2-4-5 in Figure 7.2c. With this switch, red is freed for node 0. A cd_X-path can have only one color if there is only one edge, e.g., 1-3 is a red-green path. If X has no edges with color c or d, there is no cd_X-path, e.g., node 1 has no blue-green path.

The Misra-Gries algorithm is presented as Algorithm 2. We exhibit a step-by-step execution in Figure 7.2e-o. The algorithm colors one edge with each iteration in the while loop. In Figure 7.2e, we choose $X = 0$ and $v = 1$. Then, the maximal fan only has one node v. We pick color $c =$ red (free for X) and $d =$ green (free for v). There is no cd_X-path, so there is nothing to invert. As a result, we set the color of edge (0,1) to green.

In Figure 7.2f, the maximal fan, F, of node 0 now consists of both nodes 1 and 2. We pick free colors $c =$ red and $d =$ cyan. There is still no cd_X-path. On line 8 of Algorithm 2, we pick a subfan of F such that it starts with a node, w, with free color $d =$ cyan. In our example, the subfan can be just the whole fan, so $w = 1$. Rotating the subfan sets the color of (0,2) to green. Then, we set the color of (0,1) to $d =$ cyan.

Algorithm 2 Misra-Gries Edge Coloring Algorithm

Input: Simple undirected graph $G = (V, E)$

Output: Edge coloring ϕ of G

$S \leftarrow E$
while $S \neq \emptyset$ **do**
    $(X, v) \leftarrow$ any edge in S
    $F[1 : k] \leftarrow$ a maximal fan of X with $F[k] = v$
    $c \leftarrow$ a free color of X
    $d \leftarrow$ a free color of v
    Invert cd_X-path if it exists
    Find a subfan $[F[w], F[w + 1], ..., F[k]]$ of $F[1 : k]$ such that d is free on $F[w]$
    $\phi((X, w)) \leftarrow d$
    $S \leftarrow S \setminus \{(X, v)\}$
end while

In Figure 7.2g, the maximal fan further includes node 6. We pick color $c =$ red (free for $X = 0$) and $d =$ green (free for $v = 6$). There is a cd_X-path 0-2. Thus, we invert the path resulting in setting the color of (0,2) to red, exhibited in Figure 7.2h. This is necessary since we are going to assign $d =$ green to some fan edges, so we need to free it from existing fan edges such as (0,2). Again, the subfan can be the whole fan, so $w = 1$, and we rotate the fan and set color of $(X, w) = (0, 1)$ to $d =$ green.

The subfan sometimes cannot be the whole fan. For example, a few iterations down, in Figure 7.2n, the maximal fan of $X = 4$ consists of nodes 5, 2, and 6. We pick color $c =$ gold (free for $X = 4$), and $d =$ cyan (free for $v = 6$). Since cyan is not free for node 2, the subfan we pick consists of only node 6. Therefore, there is nothing to rotate, and we set the color of (4,6) to $d =$ cyan. This results in the edge coloring in Figure 7.2o that consumes four colors, one more than the optimal solution in Figure 7.2b.

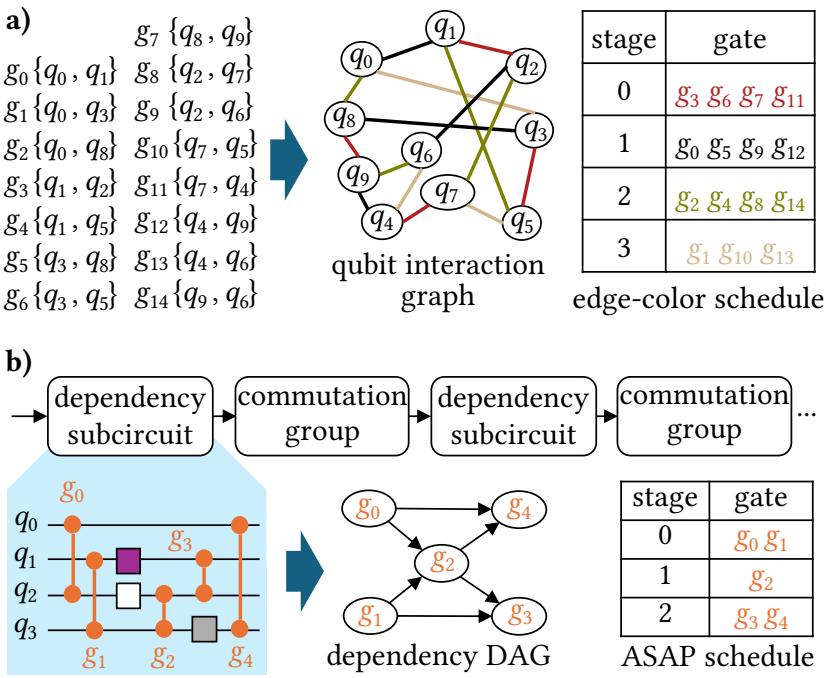


Figure 7.3: Scheduling in Enola. **a)** Scheduling a commutation group of two-qubit gates with edge coloring. **b)** Generic circuits can be considered as dependency subcircuits and commutation groups. Gates in the former is scheduled ASAP.

7.3 Scheduling: Edge Coloring

The two-qubit gates available on DPQA are controlled rotation in the Z basis, which are known to commute [LKS19]. This means a set of these two-qubit gates can be executed in any order. We can solve the scheduling of a *commutation group*, i.e., a set of commutable two-qubit gates, with the Misra-Gries algorithm, as stated in the following theorem.

Theorem 3. *For a group of commutable two-qubit gates on n qubits, suppose the optimal number of Rydberg stages to schedule these gates on DPQA is S_{opt}, there is an algorithm with time complexity $O(n^3)$ that assigns these gates to at most $S_{\text{opt}} + 1$ Rydberg stages.*

Proof. A commutation group of two-qubit gates can be represented by a *qubit interaction graph* $G = (V, E)$ where the vertices are qubits, and the edges are the two-qubit gates (Figure 7.3a). The schedule is a function $\psi : E \to \mathbb{N}$ such that a qubit can only be involved

in one gate at a Rydberg stage, i.e., for $e, e' \in E$ and $e \neq e'$, if $\psi(e) = \psi(e')$, then $e \cap e' = \emptyset$. This is contrapositive to the definition of an edge coloring, so ψ is an edge coloring. Thus, the optimal number of Rydberg stages is $S_{\text{opt}} = \chi'(G)$, which means the function Φ derived by the Misra-Gries algorithm maps the two-qubit gates to at most $S_{\text{opt}}+1$ Rydberg stages. Since $|E|$ is $O(n^2)$ where n is the number of qubits, and the Misra&Gries algorithm is $O(|V| \cdot |E|)$, the time complexity of our scheduling is $O(n^3)$. □

A more generic quantum circuit is specified by a sequence of gates. If two gates act on the same qubit, their relative order dictates a dependency. In Figure 7.3b, we exhibit an example of how one derives the dependency DAG for the two-qubit gates in a generic circuit. In this case, the scheduling problem is straightforward: the optimal number of stages is the critical path in the DAG and ASAP scheduling can achieve optimality. Although there is a way to augment the DAG to represent partially commutable circuits [IMM22], supporting this in general requires mixing logic synthesis and layout synthesis. Therefore, we make an assumption similar to [SLG19] that the whole quantum circuit is sliced into subcircuits that either respect all derived dependencies, as 'dependency subcircuits' shown in Figure 7.3b, or are commutation groups. The scheduling for the slices can be performed simultaneously and the results can be stitched together afterwards. This sliced structure is prevalent in quantum computing. An example is the graph state preparation with various applications [HDE06], which has a layer of Hadamard gates in the beginning and then commuting CZ gates. Another example is MaxCut QAOA that has alternating driver unitaries U_B with dependency and problem unitaries U_C that are commutation groups of ZZ gates.

7.4 Placement: Simulated Annealing

In placement, we map qubits to interaction sites. The two-qubit gates at each Rydberg stage thus correspond to 2-pin nets between the sites. If the nets have a long wire-length, it takes more time to move the qubits, resulting in more decoherence, the second largest

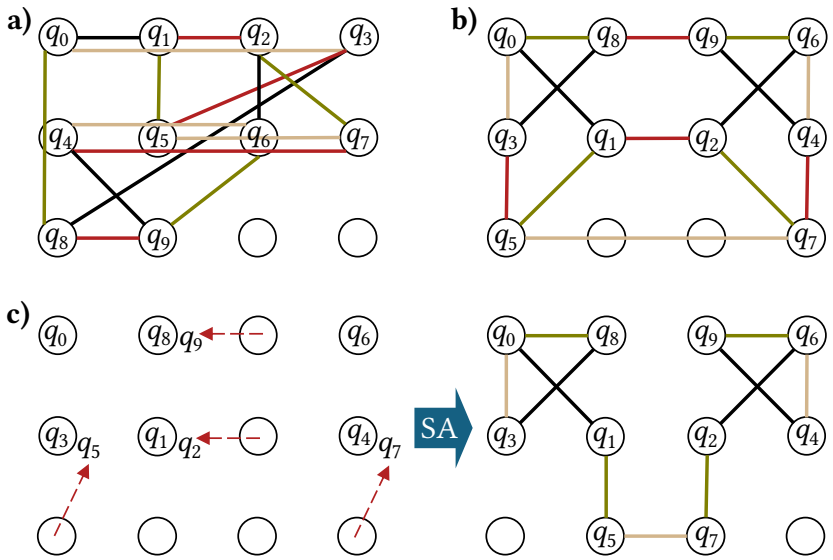


Figure 7.4: Placement in Enola. **a)** Trivial placement from left to right, from top to bottom. **b)** Placement with gate distance optimized by simulated annealing. **c)** Dynamic placement: after a Rydberg stage (red) is executed (left), run simulated annealing on moved qubits for a new placement (right).

error source. As an example, qubits can be placed trivially from left to right and from top to bottom as in Figure 7.4a. Then, the total distance of gates in the commutation group in Figure 7.3a accumulates to $25.67 \times 2.5r_b$. In comparison, an optimized placement displayed in Figure 7.4b achieves a total wire-length of $19.48 \times 2.5r_b$. To minimize the qubit traveling distances, our cost function is defined as

$$\sum\nolimits_{g(q,q')\in G} w_g \cdot \text{dist}(m(q), m(q')), \tag{7.2}$$

where w_g is the weight for gate g, m is the placement function from qubits to interaction sites, and 'dist' is the Euclidean distance.

We apply a simulated annealing algorithm, Fast-SA [CC06], to optimize this cost due to its effectiveness in solving discrete optimization problems. To enhance exploration efficiency, we confine qubits to a specific region, thus reducing the search space. Assuming the number of qubits to place is n, and the interaction sites have column indices $\{0, 1, ..., x_{\max}\}$ and

row indices $\{0, 1, ..., y_{\max}\}$, we define the chip region for exploration as $x \in [0, \max(\lfloor\sqrt{n}\rfloor + 4, x_{\max})]$ and $y \in [0, \max(\lfloor\sqrt{n}\rfloor + 4, y_{\max})]$. In Figure 7.4, $x_{\max} = 3$ and $y_{\max} = 2$. FastSA has a three-stage annealing schedule to facilitate state space exploration. At the first stage, the temperature is high. In other words, we have a higher probability to accept an inferior solution. This stage mimics a random search to explore a large solution space. Then, the second stage performs the pseudo-greedy local search with low temperature. The last stage is the hill-climbing search stage where the temperature increases again to escape from local minima. The state in the annealing process is a placement which we initialize randomly. Then, state transitions can be made by either reassigning a qubit to an empty site or exchanging the locations of two qubits. The annealing process will terminate if the temperature is lower than a threshold or the number of iterations exceed a predefined limit, so the placement algorithm has a constant runtime.

The configuration after the first Rydberg stage (red) is on the left of Figure 7.4c. The arrows indicate AOD movements from Figure 7.4b to this configuration. At this point, we can always reverse the movements to return to Figure 7.4b, and then find out the movements for the next stage (black). In this case, the placement is static for all the Rydberg stages, so we set all the gate weights to 1 in the cost function.

However, one can also consider dynamically changing the placement for the next stage. On the right of Figure 7.4c, we display a new placement where the gate between q_5 and q_7 is shorter compared to Figure 7.4b. If the placement is dynamic, gates earlier in the schedule should contribute more to the cost function. Thus, we set $w_g = \max(0.1, 1 - 0.1s_g)$, where s_g is the number of stages preceding the stage that the gate g belongs to, e.g., the gates in stages 0 to 3 will have the weights of 1, 0.9, 0.8, and 0.7, respectively. During the simulated annealing for intermediate placement, only the set of qubits necessitating relocation to vacant sites can be moved, while the remaining qubits must stay where they are. In our example, the new placement is restricted, from qubits q_2, q_5, q_7, and q_9 to the 6 empty sites. Placement of the other qubits are inherited from the previous placement. In our evaluations detailed

later, dynamic placement slightly outperforms static placement. In a commutation group, there are at most $O(n)$ stages, so our placement runtime is $O(n)$, given that each placement spends a constant time.

7.5 Routing: Independent Set

Not all AOD movements from one Rydberg stage to another can be performed simultaneously. The reason lies in the fundamental constraints of AOD: *the order of its columns cannot change, nor can the order of rows.* We consider the movements of the second stage consisting of gates (q_0, q_1), (q_3, q_8), (q_2, q_6), and (q_4, q_9), in Figure 7.5a. We define a *move* to be a 4-tuple: x and y of the source, and x and y of the destination. Since each gate has a choice of which of its two qubits to move, there are two tuples corresponding to each gate. For example, $m_0 = (0, 1, 1, 2)$ and $m_1 = (1, 2, 0, 1)$ are both for gate (q_3, q_8). We call them to be each other's *dual*. The AOD constraints are enforced by forbidding conflicts illustrated in Figure 7.5b. If the sources of two moves m and m' have the same y, i.e., $\text{src}_y(m) = \text{src}_y(m')$, the two qubits are picked up by the same AOD row. Then, $\text{dst}_y(m) = \text{dst}_y(m')$ because that AOD row can only terminate at one vertical location post-movement. Similarly, if $\text{dst}_y(m) = \text{dst}_y(m')$, then $\text{src}_y(m) = \text{src}_y(m')$. If the qubits are picked up by different rows, their relative order must be maintained, e.g., if $\text{src}_y(m) > \text{src}_y(m')$, then $\text{dst}_y(m) > \text{dst}_y(m')$. In the X direction, there are similar three types of conflicts.

These conflicts are pairwise, which means they can be encoded as edges in a graph where the vertices are the moves. We present this *conflict graph* in Figure 7.5c. A set of compatible moves constitutes an independent set (IS) of vertices. One can utilize a maximum independent set (MIS) solver for compatible moves, but MIS is NP-hard.[1] In practice, we find *maximal* independent sets are sufficient, which can be derived by 1) putting all vertices

[1]One can also imagine formulating the routing problem as a vertex coloring to find all compatible sets together, but this involves increasing the size of the graph and solving NP-hard coloring problems. Thus, we do not explore this possibility in this chapter.

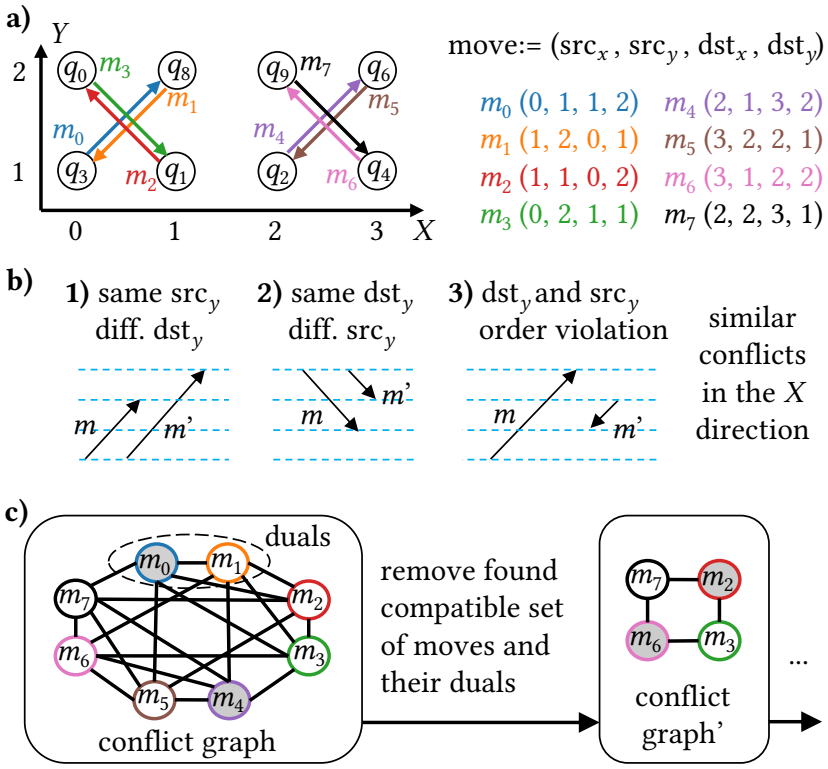


Figure 7.5: Routing in Enola. **a)** Definition of a move as a 4-tuple. **b)** Conflicts between two moves. **c)** Compatible moves are independent sets (IS) in the conflict graph (filled vertices). After finding an IS, delete the moves and their duals from the graph. The process continues until no moves are left.

in a list, 2) adding the first vertex in the list to the IS, 3) removing all its neighbors from the list, and continuing 2-3). In the first box in Figure 7.5c, assuming the list of vertices is sorted by indices, m_0 is added to the IS first, so its neighbors m_1, m_2, m_3, m_5, and m_7 are removed from the list. Next, m_4 is added to the IS and invalidates all the rest of the vertices. So, the maximal IS is $\{m_0, m_4\}$ corresponding to gates (q_3, q_8) and (q_2, q_6). Next, m_0 and m_4, along with their duals m_1 and m_5 are deleted from the conflict graph, resulting in the second box in Figure 7.5c. In the updated graph, we find the second maximal IS, $\{m_2, m_6\}$. By now, all moves are deleted, and the routing terminates.

The runtime of maximal IS is $O(|V|+|E|)$ where $|V|$ is the number of moves, which is less

or equal than the number of qubits, n. To construct the graph, we need to check conflicts for all pairs of vertices, which requires $O(n^2)$ time. The longest move in each compatible set determines the AOD movement time for this set. Thus, in our compiler, the list of moves is sorted by their distance. This sorting takes $O(n \log n)$ time. Then, the maximal IS takes $O(n^2)$ time. In summary, finding a compatible set takes $O(n^2)$ time. In the worst case, each compatible set includes only one gate. Then, we run $O(n)$ times the procedure above until all gates in one Rydberg stage are handled, resulting in $O(n^3)$ time. In total, there can be $O(n)$ Rydberg stages for a commutable group, so the total routing time is $O(n^4)$. We refer to this routing approach as *sortIS*.

To improve the runtime scaling of sortIS, we can introduce a fixed length window when scanning the list of vertices. Instead of constructing the whole conflict graph, we only construct a graph on the first K vertices in the list where K is the constant window size. These vertices are the K longest moves. Thus, both checking the conflicts between vertices and solving the maximal IS only take $O(K^2)$ time. Thus, the windowed routing takes $O(n^2 \log n + n^2 K^2)$ time. We refer to this routing approach as *windowIS*.

For each compatible set of moves, the qubits need to be picked up by the AOD and dropped off to their destination interaction sites. Turning on the AOD rows and columns and ramping up the intensity for atom transfers also takes time. To minimize this time, we need to consider the product structure of AOD, which is a research topic on its own as presented in Chapter 8. In fact, we prove that optimal routing is NP-hard from the complexity of optimizing the AOD pick-up time in Section 8.5. In practice, we do not observe this optimality to be critical to the overall fidelity. In Enola, we apply a simple approach implemented by a subroutine in OLSQ-DPQA named CodeGen where the qubits are picked up row by row. The columns may shift horizontally before picking up the next row. The CodeGen just involves scanning over all the qubits to pick up, so the runtime is less than finding the compatible sets.

7.6 Evaluation

We implemented Enola in Python and made it open source.[2] We employed KaMIS (v2.1) [HSS19] for solving the maximum independent set problems. All experiments were conducted on an AMD EPYC 7V13 64-Core Processor at 2450 MHz and 128 GB of RAM. Each fidelity data point in the figures on QAOA is an average of results corresponding to 10 randomly generated graphs of the same size.

7.6.1 Impact of Different Settings in Enola

Figure 7.6 provides the comparison of different settings in Enola on the MaxCut QAOA benchmarks. Since the scheduling is the same for all settings, the two-qubit gate fidelity term is the same. Additionally, in every setting, we use 4 atom transfers for each gate: picking up a qubit and dropping it off to the qubit it interacts with at this Rydberg stage, and the pick-up and drop-off on the way back. This means the atom transfer fidelity term is also the same for all settings. Thus, the comparison is on the decoherence term. A major improvement comes from optimizing placement, as evident by the gap between trivial placement (green triangles) and the other series. Dynamic placement (dynSA+MIS, pink cross) is slightly better than static placement (SA+MIS, blue dot). In routing, sortIS is slightly worse than MIS, as in the comparison of dynSA+sortIS (yellow star) and dynSA+MIS (pink cross). Thus, sortIS proves to be a viable replacement for MIS which is NP-hard. The windowIS method is theoretically worse than sortIS because of the limited window size. We set this size to be 1,000, larger than the scale of benchmarks in Figure 7.6. In the evaluations with larger benchmarks up to 10,000 qubits, we observe a similar number of compatible move sets and an average movement distance compared to sortIS, which means windowIS is a good heuristic to speed up the compilation.

[2] https://github.com/UCLA-VAST/Enola

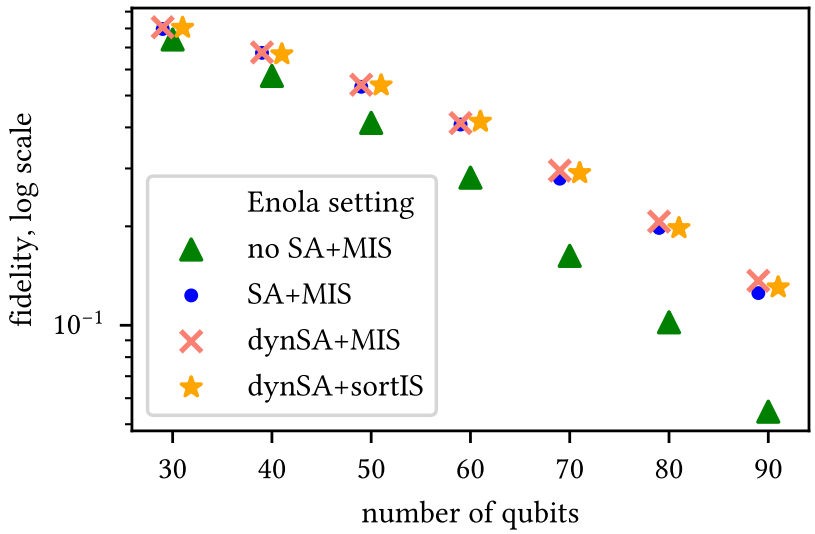


Figure 7.6: Decoherence fidelity term of different settings in Enola on 3-regular MaxCut QAOA circuits. 'no SA' means trivial placement. 'SA' means static placement. 'dynSA' means dynamic placement. 'MIS' means maximum independent using a solver. For these benchmarks, windowIS is the same with sortIS since the window size (1,000) is larger than the number of vertices in graph where we search for an IS.

7.6.2 Quality Comparison with Previous Works

In Figure 7.7, we compare Enola with OLSQ-DPQA. For dependency circuits, OLSQ-DPQA tries to execute as many gates as possible in the current front layer of the DAG, which often results in the same number of Rydberg stages as our ASAP scheduling. In some cases, like the three 'ising' benchmarks, OLSQ-DPQA suffers from elongating the critical path because its formulation cannot explore more than one rearrangement steps between Rydberg stages, which results in a notably worse fidelity compared to Enola. In some other cases like 'multiply_n13' and 'seca_n11', it appears one rearrangement step is sufficient, so the two methods produce the same number of stages. Under this scenario, OLSQ-DPQA can potentially outperform Enola because the routing in Enola is heuristic after all and may not find the optimal compatible sets of moves.

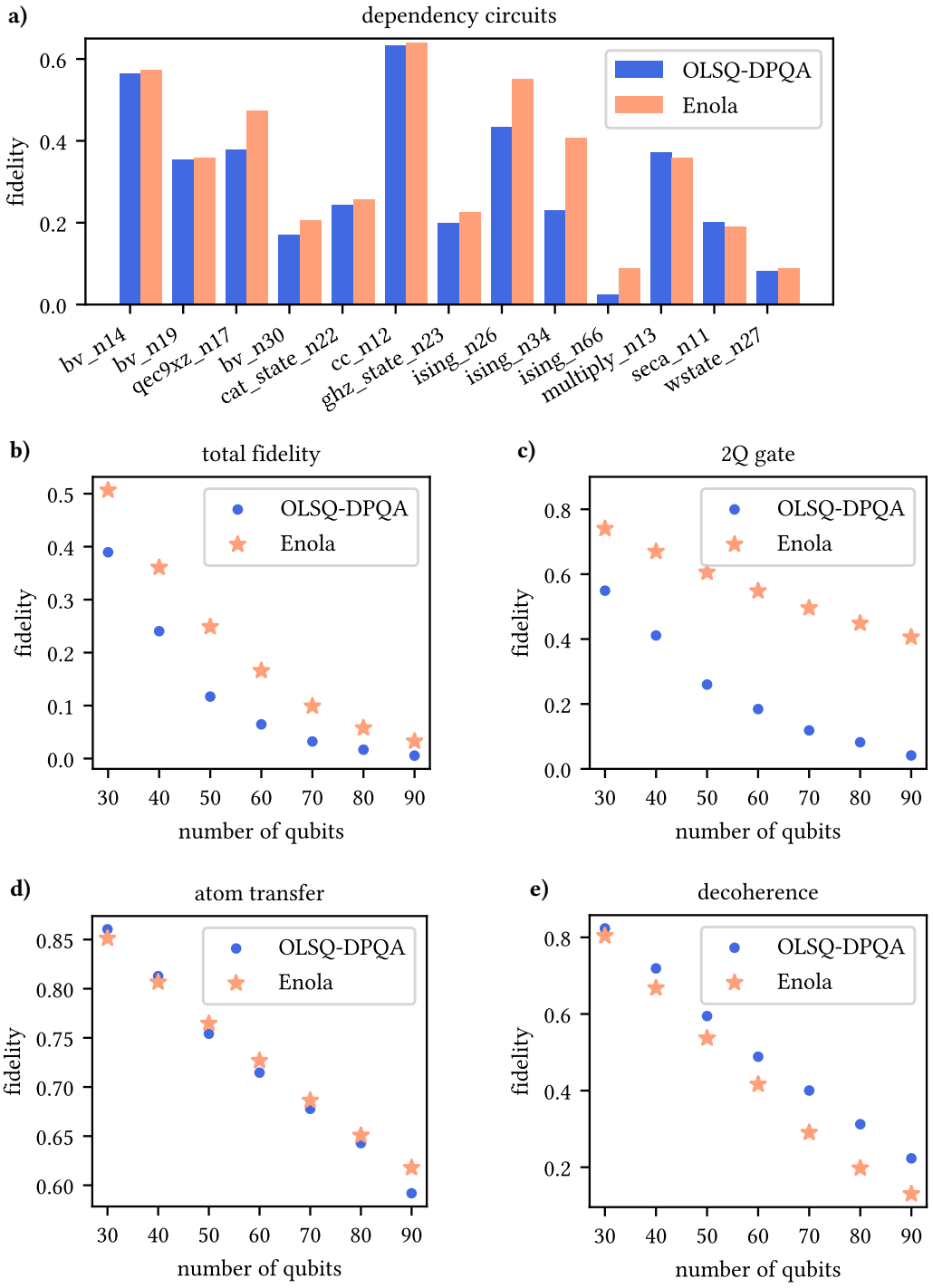


Figure 7.7: Comparison of result fidelity between Enola and OLSQ-DPQA. **a)** Comparison on dependency circuits. **b)** Comparison of total fidelity on 3-regular MaxCut QAOA circuits. **c-e)** Comparisons of different fidelity terms on the QAOA circuits.

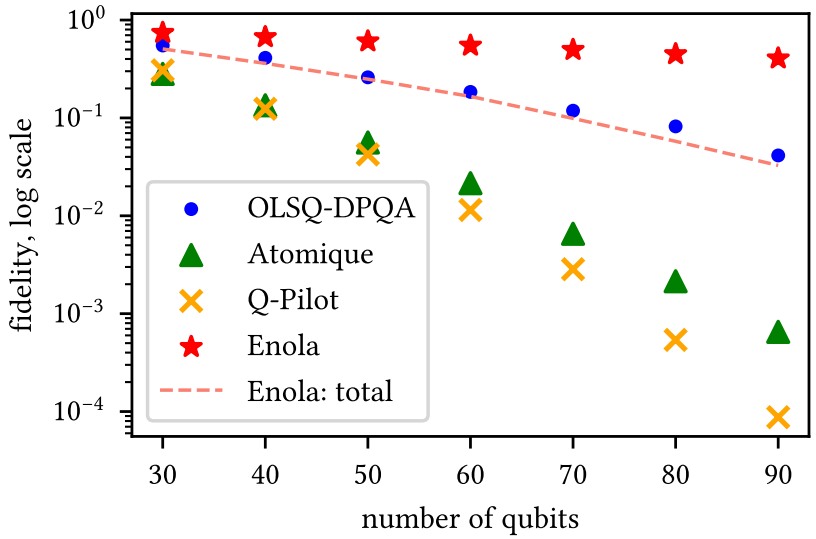


Figure 7.8: Comparison of two-qubit gate fidelity term (scattered points) on 3-regular Max-Cut QAOA circuits between Enola, OLSQ-DPQA, Atomique, and Q-Pilot. The total fidelity of Enola is also drawn for reference (dashes).

On the QAOA benchmarks, Enola clearly outperforms OLSQ-DPQA because it is able to leverage the near-optimal scheduling. We depict the comparison of overall fidelity in Figure 7.7b and the three terms in Figure 7.7c-e. In the two-qubit gate term, there is a significant gap between the two approaches. At 90 qubits, OLSQ-DPQA uses 14.6 stages on average whereas Enola only employs 4, a 3.7x reduction. In the atom transfer term, two approaches are similar, but Enola starts gaining advantage on larger benchmarks. It should be noted that in OLSQ-DPQA, atom transfers are not penalized in the SMT formulation. Examining its results with human eyes, there appears to be unnecessary transfers and movements. In Enola, we have explicit control over the transfers and movements. In the decoherence term, Enola is worse than OLSQ-DPQA. This is inevitable because we choose to prioritize the number of Rydberg stages, necessitating more AOD movements. Overall, Enola improves the fidelity by 5.9x compared to OLSQ-DPQA at 90 qubits.

The fidelity gain of Enola is even larger when compared to heuristic methods. Specifically, we participated in the development of two heuristics, Q-Pilot [WTL24] and Atomique

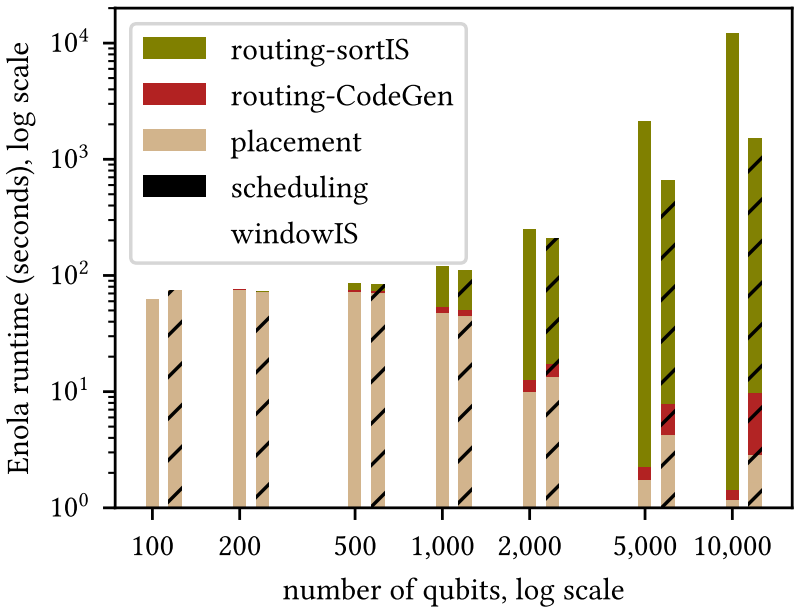


Figure 7.9: Enola runtime scaling on 3-regular MaxCut QAOA circuits. Two nearby bars correspond to the same number of qubits. The hatched bars are windowIS data (with a window size of 1,000) and the other bars are sortIS data.

[WLT24]. The former is a DPQA router that utilizes AOD only for ancilla qubits to mediate two-qubit gates between SLM qubits. It does not include a nontrivial placement solution. Atomique focuses on the placement and routes the qubits with SWAP gates. In Figure 7.8, we compare the two-qubit gate fidelity terms of all the approaches. Q-Pilot and Atomique result in more Rydberg stages than OLSQ-DPQA and Enola because the generation and recycling of the ancillas and the SWAPs require additional stages. At 90 qubits, Enola reduces the number of stages by 8.7x compared to Atomique and 10.5x compared to Q-Pilot. As a result, the two-qubit fidelity term of Enola (red star) is 779x higher than Atomique (green triangle) and 5806x higher than Q-Pilot (yellow cross). The total fidelity of Enola (dashes), including atom transfers and decoherence, is still higher than the two-qubit fidelity term of the two heuristics.

7.6.3 Runtime Scaling of Enola

Since the steps in Enola all have polynomial runtime, it is much more scalable than OLSQ-DPQA. For 3-regular MaxCut QAOA, OLSQ-DPQA can compile 90-qubit benchmarks in one day, whereas Enola compiles 100 qubit circuits with higher fidelity in a minute.

Figure 7.9 exhibits the runtime of Enola with sortIS and windowIS on larger benchmarks, up to 10,000 qubits. Note that this is a log-log plot. Different colors inside each bar provide the portion of time spent on different tasks. The scheduling is extremely fast, invisible in the plot. For benchmarks smaller than 1,000 qubits, the runtime is dominated by the placement. Although the placement scales in $O(n)$, the constant factor is large because we would like the simulated annealing to return high-quality results. Later on, the routing portion becomes dominant due to a higher asymptotic: sortIS takes $O(n^4)$ time and windowIS takes $O(n^2 \log n)$ time with a constant window size. At 10,000 qubits, the sortIS approach took 1.22E4 seconds, i.e., about 3.4 hours; the windowIS approach took 1.50E3 seconds, i.e., about 25 minutes. From the data, the runtime scaling of windowIS roughly follows $O(n^2 \log n)$: increasing the number qubits by 10x from 1,000 to 10,000, the runtime increases by 55x from 18.8 seconds to 1.04E3 seconds.

CHAPTER 8

Qubit Addressing in Dynamically Field-Programmable Qubit Arrays

In OLSQ-DPQA and Enola, we focus on parallelizing qubit movements while assuming we can always pick up a subset of qubits with AODs. Specifically, we have implemented a naïve approach in OLSQ-DPQA where the qubits to be picked up are collected from the SLM to an AOD row-by-row. A closely related problem is applying single-qubit gates with AODs to an arbitrary subset of qubits. A simple approach is, again, row-by-row [BLS24]. In this chapter, we focus on this AOD addressing problem.

As depicted in Figure 8.1a, the AOD illuminates a product of rows and columns. Quantum gates, induced by specific pulses modulated by the AOD, can address qubits at the row and column intersections. For a 2D array $X \times Y$, a *(combinatorial) rectangle* is a set of the form $X' \times Y'$, where $X' \subseteq X$ and $Y' \subseteq Y$. Specifying a rectangle requires $|X| + |Y|$ bits (one bit for each row and each column), a significant reduction compared to $|X| \cdot |Y|$ bits for all elements. This quadratic reduction is maintained while preserving individual addressability, as a single element can still be considered a rectangle.

The coarser granularity of rectangular addressing may reduce control complexity at the cost of increasing depth. A generic example problem is given by the matrix in Figure 8.1b, where the qubits to address are represented by the 1's. This matrix can be partitioned into five rectangles, each designated by distinct markers. Consecutively, each rectangle can receive the pulse to implement single-qubit gates. Minimizing the number of rectangles to partition arbitrary binary matrices becomes crucial.

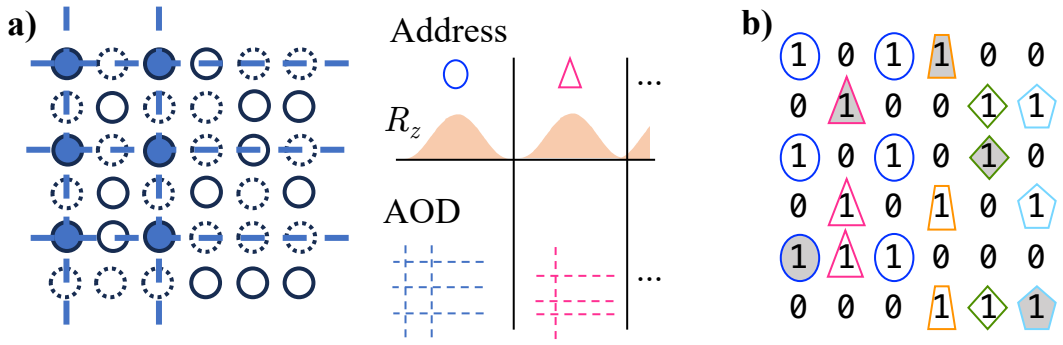


Figure 8.1: Rectangular addressing in neutral atom arrays. **a)** The experimental setup in Bluvstein et al. [BLS24]: a 2D acousto-optic deflector (AOD, blue dashes) modulates another laser to realize R_z gates on qubits at the AOD intersections (colored dots). Different qubits (uncolored dots) are addressed by changing the AOD signal. Qubits not in the pattern (dash circles) should not be addressed. **b)** Rectangular partition of a). Different markers distinguish 5 rectangles to partition the matrix. The 5 filled markers indicate a fooling set.

The rest of this chapter is organized as follows. We first review related mathematical concepts in Section 8.1. Then, we introduce our algorithms for the addressing problem in Section 8.2. The construction of benchmarks and the evaluation results are presented in Section 8.3. We discuss the problem in the context of fault-tolerant quantum computing in Section 8.4. (It may be better for the readers without prior knowledge to return to this section after reading Chapter 9.) Finally, we present an interesting proof that the routing task in Section 7.5 is NP-hard by a reduction from the qubit addressing problem.

8.1 Background

The term we have adopted, *rectangle*, is standard in communication complexity theory [Yao79], where the matrix in Figure 8.1b represents a binary function g of two variables. Alice has some i, Bob has some j, and our interest is determining the number of bits the two need to communicate to compute $g(i, j)$. If $g = 1$ uniformly on a rectangle, it is a

1-monochromatic rectangle. The number of 1-monochromatic and 0-monochromatic rectangles to partition the whole matrix serves as a crucial lower bound for the communication complexity. For an introduction to this topic, readers are referred to Kushilevitz & Nisan [KN97]. Two results they cover are worth mentioning for later discussions. First, there is an alternative definition of a rectangle:

$$(i, j) \in R \text{ and } (i', j') \in R \Rightarrow (i, j') \in R. \tag{8.1}$$

In this chapter, we only focus on 1-monochromatic rectangles, and we will refer to these as 'rectangles' from now on. The second important fact is that the partition number is lower bounded by the size of *fooling sets*. In our case, a fooling set S consists of (i, j) such that $g(i, j) = 1$, and for any distinct pair (i, j) and (i', j') in S, $g(i', j) = 0$ or $g(i, j') = 0$. Indeed, the shaded markers in Figure 8.1b identify such a fooling set of size 5, implying that our partition into 5 rectangles is optimal. Fooling sets do not guarantee a tight bound, e.g.,

$$\begin{array}{l} \text{3 rectangles are needed to partition} \\ \text{but the size of any fooling set is} \leq 2 \end{array} \quad \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{8.2}$$

The problem has a graph-theoretic interpretation when considering the matrix as the adjacency matrix of a bipartite graph, as illustrated in Figure 8.2a. The left vertices correspond to the rows, while the right vertices correspond to the columns. An edge exists between vertex i on the left and vertex j on the right if and only if element (i, j) is 1 in the matrix. Viewed in this way, a rectangle, seen as a set of edges, forms a *biclique*, i.e., a complete bipartite graph. For instance, the addressed sites in Figure 8.1a correspond to a complete (3,2)-bipartite subgraph in Figure 8.2a, as denoted by the solid edges. Therefore, the rectangular partition is equivalent to a biclique partition of a bipartite graph. Reinterpreting the left vertices as sets and right vertices as objects, the biclique partition is finding a *normal set basis* to decompose each set. In our example, the basis is $\{\{0, 2\}, \{1\}, \{3\}, \{4\}, \{5\}\}$, with the first set on the left decomposed into $\{0, 2\} \sqcup \{3\}$. The decision problem is proven NP-complete [JR93]. Even approximating the problem is NP-hard [BMB08, CHH14, CIK16].

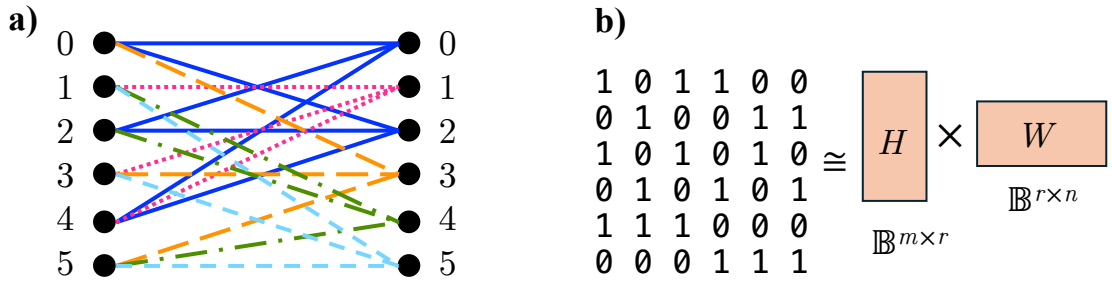


Figure 8.2: Problems equivalent to depth-optimal rectangular addressing. **a)** Interpreting the matrix as the adjacency matrix of a bipartite graph, the rectangular partition problem becomes *biclique partition* where the edges are partitioned to form complete bipartite subgraphs (different line types). **b)** Binary matrix factorization finds low-rank approximations HW of the original matrix where H and W are also required to be binary.

Amilhastre et al. [AVJ98] have characterized certain graph families where the problem can be efficiently solved.

The third perspective regarding a rectangular partition is through matrix factorization, as each rectangle precisely corresponds to a rank-1 submatrix. In Figure 8.2b, within a *binary matrix factorization* (BMF), given a binary matrix $M \in \mathbb{B}^{m\times n}$ and an integer r, the objective is to minimize $\|M - HW\|$ where $H \in \mathbb{B}^{m\times r}$ and $W \in \mathbb{B}^{r\times n}$. Note that $HW = \sum_{i=1}^{r} P_i$ where P_i is the product of column i in H and row i in W. Each $P_i \in \mathbb{B}^{m\times n}$ is 1 on a combinatorial rectangle and 0 elsewhere, so it has rank 1. The minimum r for which $M - HW = 0$ is the *binary rank*, $r_{\mathbb{B}}$ of M. In this case, $\sum_{i=1}^{r} P_i$ is an *exact* binary matrix factorization (EBMF) of M. In contrast to SVD, which provides the rank in $\mathbb{R}$, EBMF additionally requires H and W to be binary. However, it is crucial to note that the additions in the matrix multiplication in EBMF is in $\mathbb{R}$, not in $\mathbb{B}$, e.g.,

$$\begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \overset{\text{EBMF}}{\neq} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}. \tag{8.3}$$

If the addition were in $\mathbb{B}$, the equality holds. But in $\mathbb{R}$, the top-left element appears in *both* rectangles on the r.h.s., violating the disjointedness requirement of rectangular partitioning. For binary matrices, we have a straightforward lower bound [Wat16]:

$$\operatorname{rank}_{\mathbb{R}}(M) \leq r_{\mathbb{B}}(M) \quad \forall M \in \mathbb{B}^{m \times n}. \tag{8.4}$$

Zhang et al. [ZLD07] develop a BMF optimizer which is integrated into a well-known package NIMFA [ZZ12]. However, since it is not designed for EBMF but to provide approximations given a fixed r, it does not perform well for our specific purposes.

8.2 Algorithm

Given a matrix $M \in \mathbb{B}^{m \times n}$, we are interested in its exact binary matrix factorization (EBMF) $M = \sum_{i=0}^{r_{\mathbb{B}}-1} P_i$ where each $P_i \in \mathbb{B}^{m \times n}$ is 1 on a rectangle and 0 elsewhere.

Our SMT formulation encodes the problem: given M and a number b, determine if $r_{\mathbb{B}}(M) \leq b$. When $r_{\mathbb{B}}$ is unknown, we query an SMT solver with decreasing values of b to compute it. When b reaches $r_{\mathbb{B}}$, the solver should return a valid EBMF; then, when we further decrease b, the solver should output that the SMT model is unsatisfiable. Given the problem's complexity, the worst-case runtime is exponential to the size of M. Hence, the key lies in establishing relatively tight bounds for b to minimize SMT invocations.

Our approach, SAP (SMT and packing), is presented in Algorithm 3. First, our heuristic, *row packing*, provides a valid EBMF, P. Since $|P|$ is an upper bound of $r_{\mathbb{B}}(M)$, the SMT solving initiates with $b = |P| - 1$ and terminates when the SMT formula is unsatisfiable or when b falls below $\operatorname{rank}_{\mathbb{R}}(M)$, a lower bound as per Equation 8.4. P is updated each time the SMT formula is satisfiable so that it retains the best solution found thus far even if the process is prematurely interrupted.

Algorithm 3 SAP (SMT and packing) EBMF

Input: $M \in \mathbb{B}^{m \times n}$

Output: P, an EBMF of M consisting of rectangles

1: $P \leftarrow$ `row_packing_EBMF`(M) ▷ Algorithm 4
2: $b \leftarrow |P| - 1$
3: *formula* $\leftarrow$ `construct_SMT_formula`(M, b)
4: **while** $b \geq$ real rank of M **do**
5: **if** formula is satisfiable **then**
6: $P \leftarrow$ `readout_solution`(*formula*)
7: $b \leftarrow b - 1$
8: *formula* $\leftarrow$ `narrow_down_depth`(*formula*, b)
9: **else**
10: **break**
11: **end if**
12: **end while**

8.2.1 SMT Formulation

Fundamentally, we want to compute a function $f : E \to P$, where E comprises the 1's in the matrix, and P contains the rectangles. This definition offers the convenience of inherently ensuring the disjointedness of the rectangles. Furthermore, the constraints needed to enforce the validity of f in specifying rectangles can be expressed using first-order logic and equality between function values. This closely aligns with the uninterpreted function, a major addition in SMT compared to SAT [dB08, LKT23]. Concretely, the only set of constraints follows from Equation 8.1: for every pair of distinct 1's at (i, j) and (i', j'),

$$\begin{cases} f_{i,j} \neq f_{i',j'} & \text{if } M_{i,j'} = 0, \\ f_{i,j} == f_{i',j'} \;\Rightarrow\; f_{i,j} == f_{i,j'} & \text{if } M_{i,j'} = 1. \end{cases} \tag{8.5}$$

Another SMT feature we leverage is bit-vector. In fact, both the domain and range of f are bit-vectors: $f_{i,j}$ above means $f(e(i,j))$ where e is an index function of the 1's in M, and the value of f is the index of a rectangle. To narrow down the solution space as in line 8 of Algorithm 3, we just add new constraints $f_{i,j} \neq b$ for every $M_{i,j} = 1$ to the SMT formula.

8.2.2 Heuristics

A trivial upper bound of $r_{\mathbb{B}}(M)$ is the width or height of M, whichever smaller, after removing empty and duplicated rows and columns. This corresponds to partitioning the matrix into single rows or columns and consolidating duplicated ones.

The normal set basis viewpoint inspires our second heuristic. We process matrix M row by row, with the goal of forming a *basis* – each basis vector corresponds to one rectangle, as outlined in Algorithm 4. For each row r_i, as in lines 4-9, if an existing basis vector v_j is found within this row, we append i to the rectangle P_j associated with v_j. Subsequently, we remove the 1's in v_j from r_i and continue this process. The outcome is the decomposition of r_i into a disjoint union of existing basis vectors, potentially leaving a *residue* of 1's. An example is displayed in Figure 8.3a where the first four rows cannot be decomposed, so the residues are just the rows themselves, and they are added to the basis, i.e., $v_i = r_i,\ i \in \{0,1,2,3\}$. When it comes to r_4, we note it contains v_0 (circles) and v_1 (triangles), so the residue is $(0,0,0,0,1)$ denoted by the pentagon. Based on this decomposition, the rectangles P_0 and P_1, corresponding to v_0 and v_1, vertically grow to include row 4.

Since we adhere to the order of basis vectors, the decomposition can be suboptimal. For instance, we overlook the possibility of $r_4 = v_2 + v_3$, and the residue could have been 0. To mitigate this, we run the heuristic multiple times, shuffling the rows in each trial, e.g., another trial with a different row ordering is exhibited in Figure 8.3b. This is a compromise to the complexity of the problem. Formally, we are trying to find a *packing* or *exact cover* of r_i by the basis vectors, and it is an NP-complete problem [Kar72] to decide whether one exists.

Algorithm 4 Row-Packing EBMF

Input: $M \in \mathbb{B}^{m \times n}$

Output: P, an EBMF of M consisting of rectangles

$M' \leftarrow$ shuffle_rows(M)
basis $\leftarrow [\,]$; $P \leftarrow [\,]$
for $r_i \in M'$ $i = 0, 1, ..., m-1$ **do**
    for $v_j \in$ basis **do**
        if $\{1\text{'s in } v_j\} \subseteq \{1\text{'s in } r_i\}$ **then**
            $P_j \leftarrow$ `vertical_grow`(P_j, i)
            $r_i \leftarrow r_i - v_j$
        end if
    end for
    if $r_i \neq \vec{0}$ **then**
        $c \leftarrow$ `one_hot_column_vec`(i)
        for $v_k \in$ basis **do**
            if $\{1\text{'s in } r_i\} \subseteq \{1\text{'s in } v_k\}$ **then**
                $P_k \leftarrow$ `horizontal_shrink`(P_k, r_i)
                $v_k \leftarrow v_k - r_i$
                $c_k \leftarrow 1$
            end if
        end for
        basis.append(r_i)
        P.append($c \times r_i$)
    end if
end for
$P \leftarrow$ `undo_shuffle`(P, M, M')

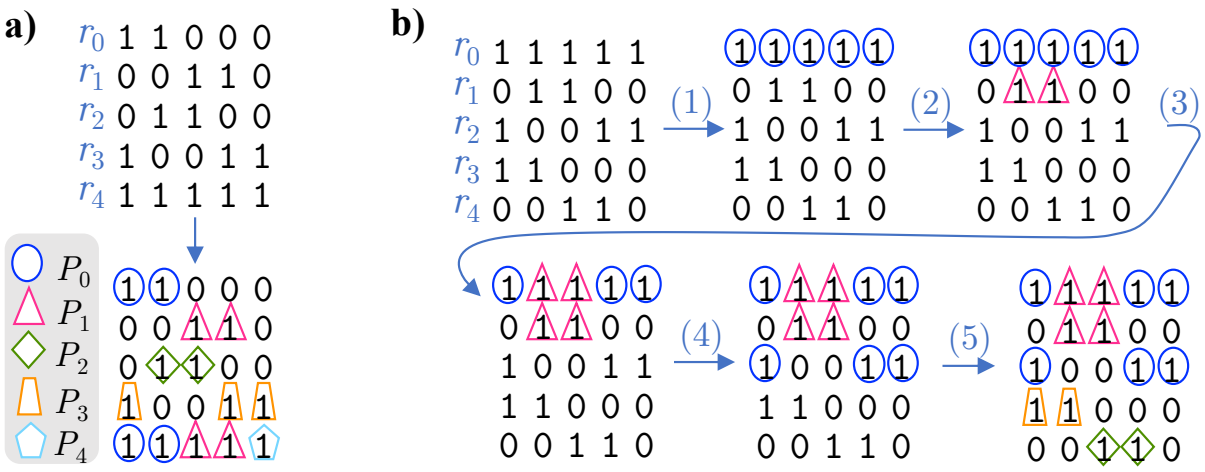


Figure 8.3: Two trials of running the row packing heuristic. Rectangles found are represented by different markers. **a)** needs 5 rectangles but **b)** needs 4.

In the presence of a residue, we perform an update to the basis in lines 11-20. The intuition behind this is that smaller basis vectors enhance the likelihood of a successful row packing. If an existing basis vector v_k contains the residue, we remove the residue from the corresponding rectangle P_k and update v_k. In step 2 of Figure 8.3b, r_1 itself (triangles) is the residue. We find existing basis vector $v_0 = r_0$ (circles) containing r_1, so v_0 is updated to $r_0 - r_1$, and then r_1 is added to the basis as v_1. Because of the updates, some existing rectangles shrink to remove the columns of 1's in the new basis vector, which we record with a column vector c. In step 3 of the example, c notes that v_0 gets updated and v_1 gets added, so the new rectangle $P_1 = c \cdot v_1$ spans rows 0 and 1, and columns 0 and 1. And the existing rectangle P_0 shrinks by removing columns 0 and 1, which leads to the successful packing of r_2 in step 4.

Finally, in line 23, we reverse the initial shuffling to derive the correct EBMF. It is worth noting that the algorithm introduces at most one rectangle for each non-repeating row, ensuring that the result is no worse than the trivial heuristic. The overall time complexity is $O(n^3k)$, where k represents the number of trials, and n denotes the larger of matrix width and

height. This complexity is due to nested loops in lines (3,4) and (3,12), with the innermost loop involving vector operations. Additionally, note that we run the heuristic on both the original matrix and its transpose, retaining the better result.

In scenarios with limited time budgets, two further compromises can be considered. The first one is removing the basis update in lines 12-18. The second one is arranging rows with a smaller number of 1's at the beginning instead of random shuffling and invoking fewer runs. Based on our experience, both of these tend to result in more suboptimal 'local minima' compared to the current setting, so we have not adopted them.

8.3 Evaluation

We implement the above approach which is open-source under the MIT License[1]. The software relies on `numpy 1.26.3` and `z3-solver 4.12.1.0` [dB08]. The evaluation is conducted on a server with an AMD EPYC 7V13 CPU and 512 GB RAM.

8.3.1 Benchmark Construction

We provide benchmarks in two sizes: 1) limiting the number of rows by 10 so that we can reliably prove the optimality of the solutions using SMT, and 2) 100×100, which is considered to be the current limit of atom array technology [BLS24].

The first benchmark set consists of random matrices. We generate 10 matrices with varying occupancies of 1's (10%, 20%, ..., 90%) for sizes 10×10, 10×20, and 10×30. For the 100×100 size, we choose occupancies of 1%, 2%, 5%, 10%, and 20%, because higher occupancies almost always result in full rank, which is trivial for our evaluation.

The second benchmark set is comprised of matrices with known optimal solutions. According to Equation 8.4, if a matrix has a k-rectangle partition and the real rank is also k,

[1] https://github.com/UCLA-VAST/EBMF

the partition is optimal. We create pairs of disjoint rows r_i and linearly independent columns c_i, leading to matrices $M = \sum_{i=1}^{k} c_i \cdot r_i$. We enforce disjointedness among the rows to ensure that the outcome matrices only contain 0's and 1's, and the rectangles cannot merge. For each rank $k = 1, 2, ..., 10$, we generate 10 benchmarks of size 10×10 with known optimal solutions.

The third benchmark set is designed to create a gap between the real rank and the binary rank. We begin by sampling a random row r and then randomly decompose it into disjoint row pairs $r = r' + r''$. The parameter for this family of benchmarks is the number of row pairs, k, which is limited to $\lfloor m/2 \rfloor$ where m is the total number of rows. The real rank of these $2k$ rows should be $k + 1$ because any pair can recover the original row, $r = r_0 + r_1$. Each pair then should provide an independent basis vector, e.g., r_{2i} for $i = 0, ..., k-1$. Note that decompositions like $r_3 = r_0 + r_1 - r_2$ require the use of negative numbers, which are not allowed in an EBMF. Consequently, the binary rank of the matrix should be larger than $k+1$. The remaining $m-2k$ rows are completed with random rows having a 50% occupancy, resulting in a total real rank equal to or slightly lower than $m - k + 1$. We generate 100 benchmarks of size 10×10 with 2, 3, 4, and 5 row pairs.

8.3.2 Results

The SMT solver allows us to compute optimal solutions. The percentage of cases achieving optimal solutions with the heuristics is presented in Table 8.1. The 'rank' column indicates the percentage of cases where the binary rank equals the real rank. Although the 100×100 benchmarks are too large for SMT to find solutions, the heuristics find solutions with the number of rectangles equal to the real rank. Consequently, these solutions are known to be optimal, and the real and binary ranks are the same. Several observations can be made.

Observation 1: the real and binary ranks are equal with high probability for random matrices. This can be attributed in part to the near-full real rank of random matrices. We observe that almost all 10×20 matrices with an occupancy of 20% and higher, all 10×30 matrices,

Table 8.1: Percentage of cases finding an optimal exact binary matrix factorization by row packing and a trivial heuristic. Row 'rank' means percentage of cases where real and binary ranks are the same. For 100×100 random benchmarks, since the heuristics managed to find optimal solutions, the real and binary ranks are the same. The SMT for these cases are too large to solve. We use SMT to compute the binary rank for all other benchmarks.

			row packing, number of trials			
benchmark	**rank**	**trivial**	**1**	**10**	**100**	**1000**
10×10, rand	98%	80%	91%	99%	100%	100%
10×20, rand	100%	100%	100%	100%	100%	100%
10×30, rand	100%	100%	100%	100%	100%	100%
100×100, rand	100%	62%	92%	96%	98%	100%
10×10, opt	100%	100%	100%	100%	100%	100%
10×10, gap, 2	74%	29%	88%	100%	100%	100%
10×10, gap, 3	63%	16%	91%	100%	100%	100%
10×10, gap, 4	47%	40%	94%	98%	99%	99%
10×10, gap, 5	42%	84%	90%	94%	96%	96%

and 100×100 matrices with an occupancy of 5% and higher are full rank, necessitating full binary rank, resulting in equality between the two.

Observation 2: the constructed benchmarks with known optimal are easy. Due to the mechanism of row packing, it can always find the optimal solutions for these benchmarks. Surprisingly, the trivial heuristic also manages to find the optimal solutions on all cases,

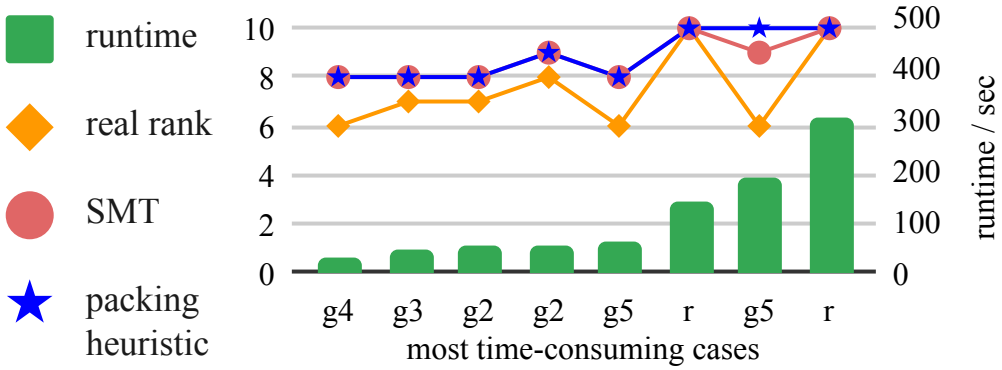


Figure 8.4: The most time-consuming cases of exact binary matrix factorization in our experiments. 'r' means it is a random benchmark, 'g2' means it comes from benchmarks with gap using 2 row pairs, etc.

because even though the row space cannot be reduced by construction, e.g.,

$$\begin{bmatrix}1\\1\\0\end{bmatrix}\begin{bmatrix}1 & 1 & 0\end{bmatrix}+\begin{bmatrix}0\\1\\1\end{bmatrix}\begin{bmatrix}0 & 0 & 1\end{bmatrix}=\begin{bmatrix}1 & 1 & 0\\1 & 1 & 1\\0 & 0 & 1\end{bmatrix},$$

the columns may be reduced by recognizing duplication.

Observation 3: row packing is an effective heuristic. On benchmarks with gaps and the large random benchmarks, there is a big gap between the trivial heuristic and even one trial of row packing, indicating row packing is highly non-trivial. As expected, the performance of row packing improves with more trials. On most of the benchmarks, it saturates at 100 trials and finds optimal solutions on a remarkable percentage of cases.

Observation 4: edge cases for row packing needs more general search. We look into the cases where row packing fails to find the optimal solution. Going through Algorithm 4, we find these cases necessitate introducing more than one basis at some rows, whereas the row packing heuristic at most introduces one new basis per row in order for efficiency.

Observation 5: the most time consuming cases are proving UNSAT. We collect the most time-consuming cases in Figure 8.4. In the majority of these cases, the SMT solver can only

find solutions with the same number of rectangles as row packing. Then, the solver goes on decreasing the bound by 1 and proves the formula to be UNSAT. This is the most time consuming task. Note that in Algorithm 3, when we terminate at any time, we can return P, the best solution found so far.

8.4 EBMF in the Context of Fault-Tolerant Quantum Computing

Fault-tolerant quantum computing performs on top of quantum error correction codes that encode each logical qubit using quantum states distributed across multiple physical qubits. A promising approach is exemplified by the surface code [FMM12], where a logical qubit manifests as a patch of physical qubits, as depicted in Figure 8.5a. For simplicity, only the data qubits are illustrated, and check qubits are not shown. A single-logical-qubit operation, designated as U, corresponds to a 2D pattern (M) of physical gates, as highlighted in the callout. On the logical level, the quantum circuit may necessitate another 2D pattern ($\hat{M}$) of logical operations. Consequently, the overall physical operation is expressed as the tensor product $\hat{M} \otimes M$. This two-level structure allows for the independent computation of the rectangular partition of $\hat{M}$ and M. Subsequently, taking the tensor product of the partitions produces the solution.

However, is this solution optimal? The real rank is multiplicative under a tensor product, as elementary row operations can be employed to make both M and $\hat{M}$ upper triangular, resulting in an upper-triangular tensor product. In contrast, whether the binary rank is multiplicative under a tensor product remains an open problem. Our aforementioned solution (tensor product of partitions) provides an upper bound: $r_{\mathbb{B}}(\hat{M} \otimes M) \leq r_{\mathbb{B}}(\hat{M}) \cdot r_{\mathbb{B}}(M)$. For lower bounds, Watson [Wat16] notes that

$$\max\left(r_{\mathbb{B}}(\hat{M}) \cdot \phi(M),\ r_{\mathbb{B}}(M) \cdot \phi(\hat{M})\right) \leq r_{\mathbb{B}}(\hat{M} \otimes M) \tag{8.6}$$

where ϕ denotes the maximum fooling set size. However, as per Equation 8.2, ϕ is not always equal to $r_{\mathbb{B}}$. In practice, the majority of M is simple, such as applying X, Z, or H to all the

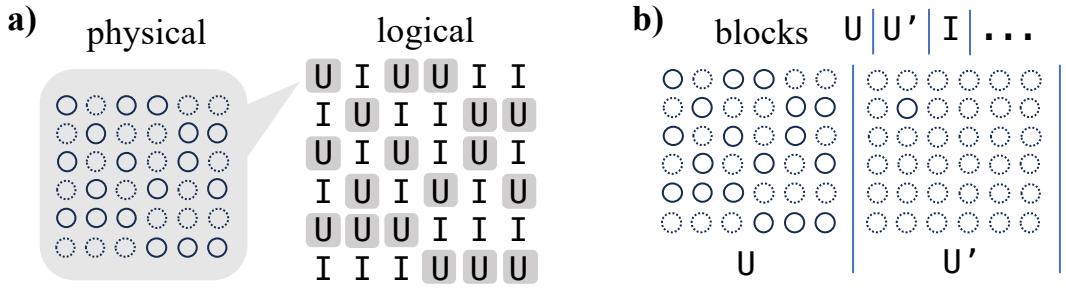


Figure 8.5: Rectangular addressing in fault-tolerant quantum computing. **a)** An operation U on 2D patterns of logical qubits can be realized by the tensor product of partitions on the logical and physical levels. **b)** For logical blocks in 1D layout and with different operations, addressing by row is usually enough.

physical qubits in one patch. In this case, all the elements of M are 1, and indeed we have $\phi(M) = r_{\mathbb{B}}(M) = 1$, so the rectangular partition of $\hat{M}$ leads to an optimal solution.

Another family of quantum error correction codes gaining popularity is quantum low-density parity-check codes, which can take advantage of the mobility of atom arrays [XBP24] In this code, logical qubits are more globalized, with multiple logical qubits stored in one *logical block* instead of one qubit per block. These blocks are usually arranged in a 1D fashion, as shown in Figure 8.5b, because they only serve as memory, and logical qubits need to be read out to a computing zone. Considering logical operations that can be realized with single-qubit gates in this setting, the pattern on each block can be quite different, depending on the offset of logical qubits inside the blocks. We conjecture that addressing qubits row by row is usually sufficient in this case, as in our evaluation, we find that given the same occupancy, the 10×20 and 10×30 random matrices are much easier to be full rank than the 10×10 matrices.

8.5 NP-Hardness Result of the DPQA Routing Problem

In the DPQA routing problem (Section 7.5), we are given a set of pairwise disjoint two-qubit gates and the qubit placement, i.e., a map from qubits to interaction sites. The goal is to generate compatible sets of moves to realize all the two-qubit gates. By our assumption, each move concerns one and only one gate, so the number of atom transfers is fixed regardless of the routing strategy. Thus, different routing strategies have the same two-qubit gate fidelity and atom transfer fidelity but may have different decoherence fidelity due to different time spent. An optimal routing solution should minimize both the time of AOD movements and that of atom transfers. In the main text, we focus on the former. To minimize the atom transfer time, we need to consider which qubits can be picked up in parallel.

Because of the product structure of the AOD, it can transfer qubits locating at a rectangle of SLM traps, e.g., in Figure 2.4a, it aligns with 2-by-2 traps and picks up 3 qubits. If more qubits need to be collected, we can slightly shift the existing AOD rows and columns, and ramp up some other rows and columns which, again, will pick up qubits in a rectangle of SLM traps. For example, in Figure 8.6a, 1's like M_{00} means the qubit is to be collected, whereas 0's like M_{13} means the qubit should not be collected. The 5 rectangles to partition the matrix incur 5 parallel pick-ups. Since each parallel pick-up takes a constant time T_{trans}, we would like to minimize the number of rectangles. Below, we prove that the depth-optimal rectangular addressing problem is reducible to the DPQA routing problem. The idea is that in the constructed routing problem, there is only one compatible set of moves, so an optimal routing solution collects qubits with a minimum number of parallel pick-ups, i.e., rectangles.

Theorem 4. *The DPQA routing problem is NP-hard.*

Proof. Given a matrix $M \in \mathbb{B}^{m\times n}$, e.g., Figure 8.6a, for which we want to compute the optimal rectangular partition, we construct a DPQA routing problem. There are $2mn$ qubits in the routing problem. The qubit placement is from left to right and from top to bottom. Each row contains $2n$ qubits as illustrated in Figure 8.6b. The black dots are the qubits with

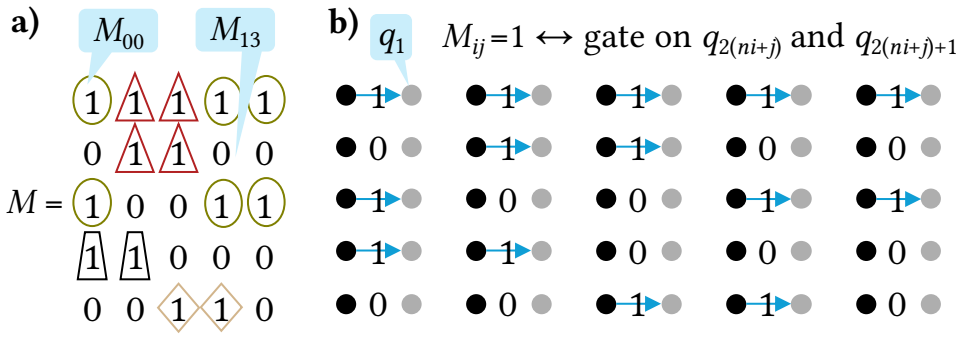


Figure 8.6: NP-hardness of DPQA routing. **a)** A rectangular partitioning problem. **b)** Reducing a) to a DPQA routing problem.

even indices whereas the gray dots are the qubits with odd indices. The set of two-qubit gates to route is $\{(q_{2(ni+j)}, q_{2(ni+j)+1}) \mid 1 \leq i \leq m,\ 1 \leq j \leq n,\ M_{ij} = 1\}$.

Given the qubit placement, the movements for all the gates can be performed simultaneously by shifting all the black dots at the beginning of all the arrows to the right by a single qubit separation. Since this move is as parallel as possible and as small as possible, any optimal routing solution must move the qubits this way. Then, the fidelity only differs in how the qubits are picked up. An optimal routing solution must provide the minimum number of parallel pick-ups to collect qubits corresponding to the 1's in M. Since each parallel pick-up corresponds to a rectangle in M, the optimal routing solution computes exactly the optimal rectangular partitioning of M. Thus, we have reduced an NP-hard problem, optimal rectangular partitioning, to a DPQA routing problem, which means the DPQA routing problem is NP-hard. □

CHAPTER 9

Introduction to Quantum Error Correction and Fault-Tolerant Quantum Computing

In this section, we build up the background knowledge necessary to comprehend our contributions for fault-tolerant quantum architectures. We begin by covering some essential mathematical concepts and then explain key aspects of quantum error correction and fault-tolerant quantum computing, using a surface-code architecture as the running example.

9.1 Mathematical Background

To lay the groundwork for understanding quantum error correction, we introduce several important mathematical concepts. The Pauli operators are fundamental in quantum computing, serving as the model for quantum errors and playing a crucial role in quantum information. The stabilizer formalism allows for the description of quantum states using a set of Pauli operators instead of state vectors, which can have exponentially many non-zero amplitudes, simplifying the representation and manipulation of some important quantum states. Clifford matrices are operators that transform within the Pauli group, essential for the construction and manipulation of quantum states and particularly important in quantum error correction. Lastly, the ZX calculus is a graphical mathematical language that offers a convenient alternative to the traditional circuit picture for performing algebra related to quantum error correction, providing clearer insights and simplifications through its visual approach.

9.1.1 Pauli Group

The n-qubit Pauli group $\mathbf{P}_n$ is defined as $\mathbf{P}_n = \{i^{j_0}\sigma_{j_1} \otimes \ldots \otimes \sigma_{j_n} \mid j_0, j_1, \ldots, j_n = 0, 1, 2, 3\}$, where the σs are Pauli matrices:

$$\sigma_0 = I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \sigma_1 = X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \sigma_2 = Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad \sigma_3 = Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{9.1}$$

$\mathbf{P}_n$ consists of 4^{n+1} elements since each of the j indices has four choices and the constant factor can be ± 1 or $\pm i$. The group properties of $\mathbf{P}_n$ are verified by:

1. Matrix multiplication is associative.
2. $I \otimes I \otimes \ldots \otimes I$ serves as the identity element.
3. The inverse of each element $i^{j_0}\sigma_{j_1} \otimes \ldots \otimes \sigma_{j_n}$ is given by $i^{4-j_0}\sigma_{j_1} \otimes \ldots \otimes \sigma_{j_n}$, which is also in $\mathbf{P}_n$.

Elements of $\mathbf{P}_n$ are sometimes referred to as *paulistrings*, as they can be represented as a string of length $n+1$ where the first entry is the constant factor and the following n characters are either I, X, Y, or Z. The *weight* of a paulistring is the number of characters that are not I.

The commutator is defined as $[A, B] := AB - BA$. If $[A, B] = 0$, the two operators commute. Conversely, the anticommutator is defined as $\{A, B\} := AB + BA$. If $\{A, B\} = 0$, the two operators anticommute. The commutator and anticommutator of single-qubit Pauli operators are detailed in Table 9.1, showing that *two Pauli operators either commute or anticommute.*

Considering multiple qubits, the product of two n-qubit Pauli operators $(i^{j_0}\sigma_{j_1} \otimes \ldots \otimes \sigma_{j_n})$ and $(i^{j_0'}\sigma_{j_1'} \otimes \ldots \otimes \sigma_{j_n'})$ is $i^{j_0+j_0'}(\sigma_{j_1} \cdot \sigma_{j_1'}) \otimes \ldots \otimes (\sigma_{j_n} \cdot \sigma_{j_n'})$. Thus, it is straightforward to verify that two multi-qubit Pauli operators either commute or anticommute depending on the parity of the number of anticommutes among their corresponding single-qubit Pauli

Table 9.1: Commutators and anticommutators of Pauli matrices. For each entry at row P and column P', the first number is $[P, P']$ and the second number is $\{P, P'\}$.

$[\cdot,\cdot],\ \{\cdot,\cdot\}$	I	X	Y	Z
I	$0, 2I$	$0, 2X$	$0, 2Y$	$0, 2Z$
X	$0, 2X$	$0, 2I$	$2iZ, 0$	$-2iY, 0$
Y	$0, 2Y$	$-2iZ, 0$	$0, 2I$	$2iX, 0$
Z	$0, 2Z$	$2iY, 0$	$-2iX, 0$	$0, 2I$

operators. Specifically, by comparing characters in the two paulistrings and counting the occurrences where they are different and neither is I, we find that if there is an even number of such occurrences, the two paulistrings commute; otherwise, they anticommute. For example, $I_1X_2I_3X_4$ and $Z_1Z_2Z_3Z_4$ commute since X meets Z twice at qubits 2 and 4; $IXYZ$ and $ZXYY$ anticommute since Z meets Y once.

Pauli operators with constant ± 1 are *involutory*, i.e., $P^2 = I$, which means it can only have eigenvalues ± 1. The projector to the eigenspaces of ± 1 are $(I \pm P)/2$. If a pauli has non-zero weight, i.e., it does not solely consist of I, then it is traceless. This property arises because at least one component $P_i = X$, Y, or Z, which is traceless, as demonstrated in Equation 9.1. Consequently, the trace of the operator can be calculated as follows:

$$\mathrm{Tr}(P_1 \otimes \ldots \otimes P_n) = \mathrm{Tr}\ P_i \cdot \prod_{j \neq i} \mathrm{Tr}\ P_j = 0. \tag{9.2}$$

This results from the multiplicative property of traces across tensor products. Since trace is equal to the sum of eigenvalues counted with multiplicities, if a Pauli operator is traceless, it has the same number of eigenvalues $+1$ and -1. Thus, ± 1 eigenspaces of a positive-weight and involutory Pauli operator have the same dimension, which is half of the dimension of the whole state space.

9.1.2 Stabilizer Group

For a quantum state $|\psi\rangle$, a stabilizer is any operator P such that $P|\psi\rangle = |\psi\rangle$. For instance, X is a stabilizer of $|+\rangle$, and $-X$ is a stabilizer of $|-\rangle$. Notably, we focus on states that are stabilized by Pauli operators, so henceforth, the stabilizers we refer to are paulis. If $|\psi\rangle$ is a multi-qubit state, it possesses more than one stabilizer. For example, the Bell state $|\Phi^+\rangle := (|00\rangle + |11\rangle)/\sqrt{2}$ has stabilizers XX and ZZ.

Stabilizers of a state must commute; otherwise, by the properties of the Pauli group, P and P' anticommute, leading to $PP'|\psi\rangle = -P'P|\psi\rangle$. However, since both P and P' are stabilizers, we have $PP'|\psi\rangle = P|\psi\rangle = |\psi\rangle$ and $-P'P|\psi\rangle = -P'|\psi\rangle = -|\psi\rangle$, which implies $|\psi\rangle = 0$ — a contradiction as $|\psi\rangle$ is not a quantum state. (Note that this zero is not $|0\rangle$ but zero in the vector space of states.)

A stabilizer group, S, is defined as an abelian subgroup of the Pauli group where $-I \notin S$ to avoid trivial cases: if $-I|\psi\rangle = |\psi\rangle$, then $|\psi\rangle = 0$. Moreover, the constants of the paulistrings in a stabilizer group must be ± 1, excluding $\pm i$ since the square of such a paulistring would yield $-I$. This means the paulis in a stabilizer group are involutory. The stabilizer group corresponding to $|\Phi^+\rangle$ includes $\{II, XX, ZZ, XX \cdot ZZ = (-iY) \cdot (-iY) = -YY\}$. Each stabilizer group has a set of *generators*; for $|\Phi^+\rangle$, these are $\{XX, ZZ\}$. No generator can be expressed as a product of other generators, thus although the all-identity pauli is in any stabilizer group, it is not a generator. Therefore, the stabilizer group generators can only be positive-weight and involutory paulis.

Typically, a stabilizer group is represented by its generators, e.g., $\langle XX, ZZ\rangle$, and sometimes the term "stabilizers" actually refers to a set of generators in literature. If there are s generators, the stabilizer group contains 2^s elements.

A stabilizer group S does not necessarily correspond to a single state but a subspace of the state space. Each generator effectively 'cuts' the state space in half. To illustrate, consider adding generators one by one to the generating set. Initially, an empty set corresponds to the

entire state space of dimension 2^n. Adding the first generator, g_1, projects the state space to its $+1$ eigenspace, $(I + g_1)/2$, halving its dimension to 2^{n-1}. Then, $\text{Tr}[g_2(I + g_1)/2] = [\text{Tr}(g_2) + \text{Tr}(g_2 g_1)]/2 = 0$, so the ± 1 eigenspaces of g_2 have the same dimension inside the $+1$ eigenspace of g_1. Note that $\text{Tr}(g_2 g_1) = 0$ is due to the fact that $g_2 g_1$ is in the stabilizer group and cannot be I which would imply $g_2 = g_1$. Subsequent generators similarly divide the remaining space. Thus, if there are m generators, the subspace has dimension 2^{n-m}.

The benefit of the stabilizer formalism is its efficiency: it avoids the need to specify the amplitudes of all basis states, which can be exponentially many. To represent a state, we require only $n(2n + 1)$ bits: there are n generators, each one needs one bit for the sign and 2 bits for the choice of I, X, Y, or Z for each of the n qubits. Note that this efficiency is only for specific states that we are interested in, which can be stabilized by paulis.

9.1.3 Clifford Group

The n-qubit Clifford group consists of unitary operators that normalize the Pauli group. For any Clifford operator C, it holds that $CPC^\dagger \in \mathbf{P}_n$ for any $P \in \mathbf{P}_n$. This implies that the map defined by conjugation with a Clifford maps paulis to paulis. Therefore, instead of explicitly writing out the matrix of a Clifford C, we can represent it with stabilizer flows $P \to CPC^\dagger$ [MBG23]. If the state is stabilized by P before the gate C, it is stabilized by $CPC^\dagger$ after the gate.

To fully characterize a Clifford gate C, we can derive $2n$ flows where the inputs are $ZI...I$, $XI...I$, $IZ...I$, $IX...I$, ..., $I...IZ$, $I...IX$. Alternative sets of paulistrings could be chosen, but this "canonical" set is the simplest and sufficient because any input paulistring can decompose as a product of these canonical inputs up to a constant. The stabilizer flows for some common gates are displayed in Figure 9.1, where the red letters represent inputs, and the blue letters represent outputs. For instance, the flows for the CNOT gate are $ZI \to ZI$, $XI \to XX$, $IZ \to ZZ$, $IX \to IX$, verifiable via the linear algebra calculations. Intuitively, an X flips a qubit, so if it is on the control qubit before a CNOT, it should propagate to

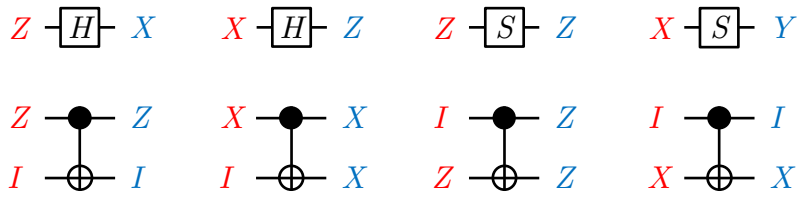


Figure 9.1: Stabilizer flows of representative Clifford gates.

the target; if it is on the target before the CNOT, it should not propagate to the control. As mentioned in Section 1.1.2, the roles of control and target are reversed when we view the CNOT in the Hadamard basis, so a Z propagates from the target to the control through a CNOT, but a Z does not propagate from the control to the target.

The stabilizer flows of a gate form a group. The multiplication between two elements is performed by multiplying the inputs and outputs separately:

$$[P \to CPC^\dagger] \cdot [P' \to CP'C^\dagger] := [PP' \to CPC^\dagger \cdot CP'C^\dagger] = [PP' \to CPP'C^\dagger]. \tag{9.3}$$

This operation is associative. The identity element is $I \to I$. The inverse of $P \to CPC^\dagger$ is $P^\dagger \to CP^\dagger C^\dagger$. To construct the output for any input, we can decompose the arbitrary input into our canonical inputs, e.g., $iYX = ZI \cdot XI \cdot IX$, and then multiply the corresponding outputs, e.g., for the CNOT, the output for iYX is $ZI \cdot XX \cdot IX = iYI$.

The Clifford group can be generated by three gates: H, S, and CNOT [NC10]. Therefore, an n-qubit Clifford circuit composed of these gates can be characterized by tracking the change of $2n$ paulistrings through each gate according to Figure 9.1. This forms the basis for why Clifford circuits can be efficiently simulated by classical computers [AG04]. Conversely, non-Clifford gates, like the T gate, cannot be characterized simply by paulistrings:

$$TXT^\dagger = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ 0 & e^{-i\pi/4} \end{pmatrix} = \begin{pmatrix} 0 & e^{-i\pi/4} \\ e^{i\pi/4} & 0 \end{pmatrix}, \tag{9.4}$$

which is not a member of the Pauli group. Indeed, the inclusion of any non-Clifford gate alongside Clifford gates is sufficient for universal quantum computing [NC10].

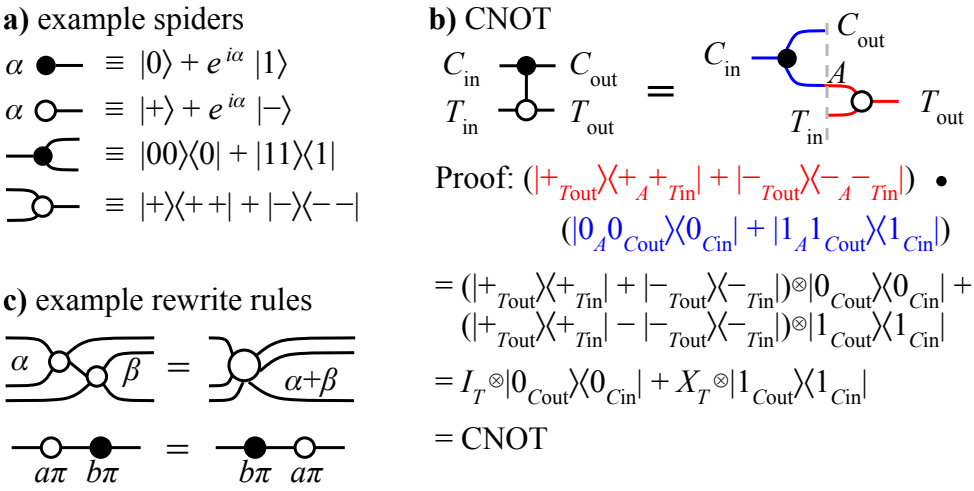


Figure 9.2: ZX calculus. **a)** Examples of spiders. **b)** Proof of the CNOT diagram. **c)** Examples of rewrite rules: merge phases and exchange whole-π spiders.

9.1.4 ZX Calculus

ZX calculus is a graphical language that facilitates intuitive algebraic manipulations related to quantum computing. Below, we introduce only a small subset of the ZX calculus. Interested readers are encouraged to consult [Wet20] for a more comprehensive treatment.

As exhibited in Figure 9.2a, the basic components in ZX diagrams are two types of *spiders*, Z-spider (solid dots) and X-spider (circle). These spiders are connected by wires. (Our color scheme may deviate from existing literatures: Z-spiders and X-spiders are green/light and red/dark, respectively, in [BH20] and [Wet20].) Each spider has a parameter named *phase*, which is an angle in $[0, 2\pi)$. If the phase is not annotated, it is assumed 0, like the lower two spiders in Figure 9.2a.

Fundamentally, each spider is a tensor, and a wire between two spiders is a tensor contraction. Thus, what matters are the labeling of open wires, the type and phase of spiders, and their connectivity. In contrast, the geometric locations of spiders are irrelevant, and the wires can bend and stretch. In Figure 9.2b, two equivalent ZX diagrams of CNOT are displayed, up to these meaning-preserving deformations. We can apply the definition of

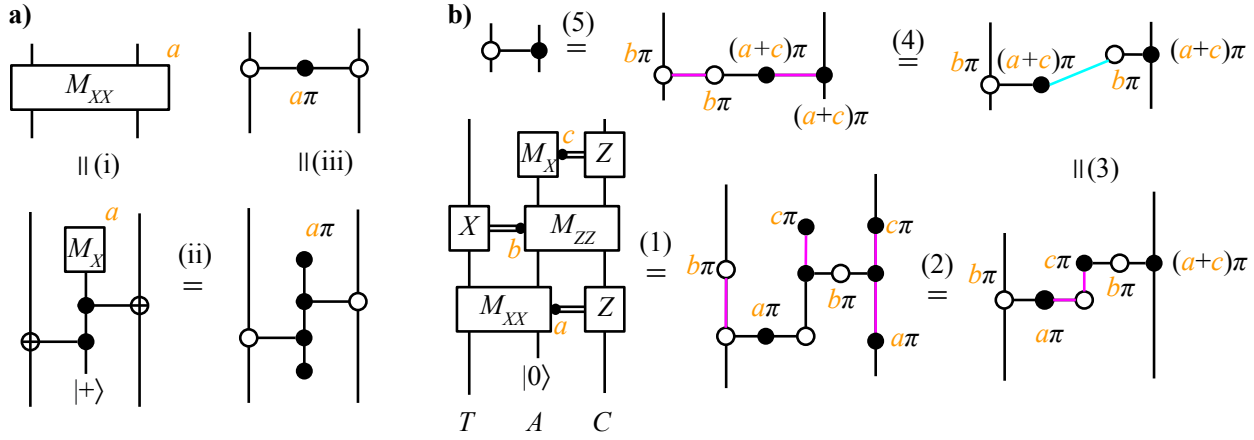


Figure 9.3: ZX derivation including measurements and feed-forward gates. **a)** ZX diagram of XX parity measurement. Yellow numbers are measurement results, 0 or 1. **b)** Proof of the CNOT circuit using parity measurements. Magenta means applying the spider merging rule; cyan means applying the whole-π spider exchange rule.

spiders in the diagram on the right to prove it is indeed a CNOT. The first step is due to the contraction at point A: $\langle\pm|0\rangle = 1$ and $\langle\pm|1\rangle = \pm 1$. (We ignore the constant $1/\sqrt{2}$ here.) The second step is by the definition of Pauli operators: $I = |+\rangle\langle+| + |-\rangle\langle-|$ and $X = |+\rangle\langle+| - |-\rangle\langle-|$. The final step is the definition of a CNOT: in case of $|0\rangle$ on the control, do nothing; in case of $|1\rangle$ on the control, apply X to the target.

Based on the algebra of spiders, some rewrite rules are derived to help one manipulate ZX diagrams, e.g., merging same-type spiders, and exchanging whole-π phase spiders ($a, b = 0, 1$), as in Figure 9.2c. We refrain from more details about rewrite rules since they are irrelevant at the moment.

As an application, we prove the circuit identity depicted in Figure 1.2e. The ZX diagram for the XX parity measurement is derived in Figure 9.3a, where a represents the result of the measurement. The derivation holds whether $a = 0$ or 1.

- In step (i), we expand the parity measurement circuit as shown in Figure 1.2d, with time progressing from bottom to top.

- In step (ii), we translate the circuit into a ZX diagram using Figure 9.2.
- In step (iii), we apply the spider merging rule to simplify the diagram.

For a ZZ parity measurement, as opposed to XX, we simply reverse the colors of the spiders. This can be proved in a similar manner.

Leveraging these results, we translate the parity measurement version of a CNOT in step (1) of Figure 9.3b, where C denotes the control qubit, T the target qubit, and A the ancilla. Each feed-forward single-qubit gate is associated with the same integer variable (a, b, or c) as the corresponding measurement.

- In steps (2) and (3), we apply the spider merging rule on wires highlighted in magenta.
- In step (4), we use the whole-π exchange rule as highlighted in cyan.
- In step (5), we apply the merging rule again to arrive at the ZX diagram for a CNOT.

While we could certainly prove this identity using the circuit language as seen in Equation 1.14, the calculation is considerably more complex than using ZX calculus.

9.2 Quantum Error Correction with Surface Codes

In classical error correction, redundancy is introduced to protect data. Similarly, quantum error correction uses multiple physical qubits to encode a logical qubit. In this section, we use surface codes as an example to explain the construction of quantum codes and how they can detect quantum errors.

9.2.1 Code Construction

The majority of quantum error correcting codes are *stabilizer codes* [Got97]. For a code with n physical qubits and a stabilizer group $S = \langle g_1, g_2, \ldots, g_m \rangle$, the logical subspace is stabilized

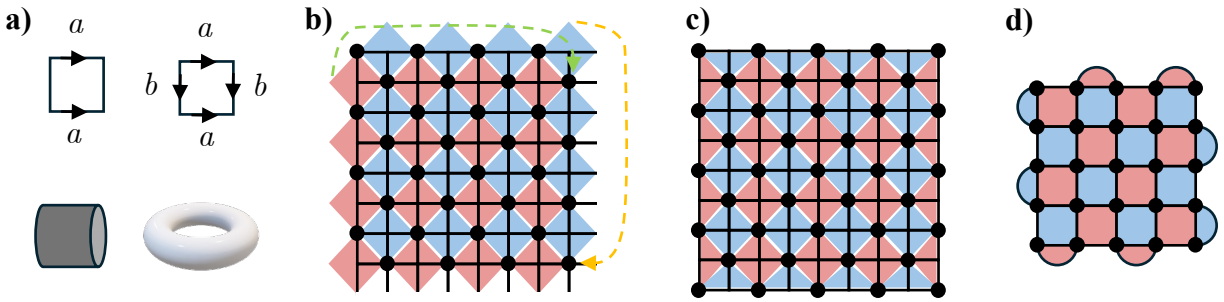


Figure 9.4: Construction of **a,b)** Kitaev's torus code, **c)** the surface code, and **d)** the rotated surface code. Red and blue means X and Z stabilizers. Dots are data qubits.

by S. The number of logical qubits in this code, k, is given by $k = n - m$. Among stabilizer codes, many are *CSS codes* [CS96, Ste96], where the generators of S can be chosen to consist only of purely-X paulistrings and purely-Z paulistrings. For visual clarity, we draw X-type and Z-type stabilizers in different colors, red and blue, respectively.

Three closely related codes are provided in Figure 9.4. The *torus code* (Figure 9.4b) is constructed by assigning a square grid on a torus and tessellating four-body X and Z stabilizers on the torus. To comprehend this, consider the example in Figure 9.4a, where identifying certain boundaries constructs complex surfaces like a torus from a plane surface. Identifying two opposite sides (a) of a square results in a pipe; further identifying two opposite sides (b) yields a torus. The two dashed arrows in Figure 9.4b indicate the identification of opposite sides of the grid. Approximately half of the intersection points of the grid are data qubits (represented by dots).

The torus code is scalable since the granularity of the underlying grid can be adjusted to change the number of qubits. In the specific example shown, there are 32 data qubits. Each stabilizer generator in the torus code acts on four qubits. Stabilizers on the boundaries roll over to the opposite side due to the identified boundaries. In Figure 9.4b, we illustrate 16 $ZZZZ$ stabilizers and 16 $XXXX$ stabilizers. However, one $ZZZZ$ stabilizer is redundant because multiplying all 16 stabilizers together counts each data qubit exactly twice,

meaning the product is always the identity. Thus, only 15 out of the 16 Z-type stabilizers are independent. Similarly, one $XXXX$ stabilizer is redundant. Therefore, there are 30 independent stabilizer generators compared to 32 qubits, resulting in 2 logical qubits.

The *surface code*, displayed in Figure 9.4c, is very similar to the torus code, except that the boundaries are not identified and instead, three-body stabilizers are introduced at the boundaries. In this specific example, there are 41 data qubits, 20 independent X-type stabilizers, and 20 independent Z-type stabilizers, resulting in one logical qubit. A more qubit-efficient version is the *rotated surface code*, shown in Figure 9.4d. Henceforth, when referring to the surface code, we typically mean the rotated version. It appears as a diamond cut from the center of Figure 9.4c and then rotated by 45 degrees. In this example, each grid intersection is a data qubit. With 25 data qubits, 12 independent X-type stabilizers, and 12 independent Z-type stabilizers, there is one logical qubit.

9.2.2 Decoding Errors

The nature of quantum errors is a very involved topic. They are certainly not just bit flips. For an in-depth discussion, readers are referred to Chapter 10 of [NC10] or other relevant literature. In this dissertation, we adopt the standard quantum error model where errors to detect are modeled as paulistrings, with lower-weight paulistrings being more probable.

Suppose a quantum state $|\psi\rangle$ is stabilized by $g_1, g_2, \ldots, g_m$. To protect $|\psi\rangle$, we periodically measure the stabilizer generators, a process known as *syndrome extraction*. The results of these measurements, termed the syndrome, help identify the presence of errors. An error E, being a paulistring, either commutes or anticommutes with each stabilizer. If E anticommutes with a stabilizer g_i, then $g_i E|\psi\rangle = -E g_i|\psi\rangle = -E|\psi\rangle$, placing $E|\psi\rangle$ in the -1 eigenspace of g_i, which in turn flips the outcome when g_i is measured.

The goal of quantum code design is to ensure that all potentially damaging errors with a weight less than a certain threshold, named the *code distance* d, are detectable. An error

E that commutes with every stabilizer goes undetected. However, a lot of errors in this type are non-damaging because *they are stabilizers*, which do not alter the logical quantum state. For example, the surface code may include stabilizers with weights as low as two. In this case, $E|\psi\rangle = |\psi\rangle$ since the error E is a stabilizer. However, there are damaging errors that commute with all stabilizers, such as the logical X operator, which is an X on five data qubits shown in Figure 9.5a. Such errors are undetectable yet alter the logical state by applying a logical X gate. If the code distance is d, then there are only such errors with weight $\geq d$.

With CSS codes, we can detect X errors using Z-type stabilizer measurements and Z errors using X-type stabilizer measurements. This is because an error can push the state outside the $+1$ eigenspace of some stabilizers, which can be detected with the syndromes. A Y error, affecting both X and Z components, is detected by both types of checks. In Figure 9.5b, we present Z checks as dots.

Detecting X errors is formulated as a minimum-weight perfect matching on a rotated grid graph connecting the Z checks, represented by dashes, as shown in [DKL02]. Additional *boundary nodes*, represented as diamonds and not part of the stabilizers, are included to ensure that every edge in the decoding graph corresponds to a data qubit.

Figure 9.5c-f provides examples of decoding scenarios. A single X error, as shown in Figure 9.5c, flips two Z checks it touches, because they both anticommute with the error. The minimum-weight perfect matching connects these detection events, correctly identifying the qubit with the error. Figure 9.5d demonstrates a weight-2 error, setting off two detection events. The error correction algorithm matches these events to the corresponding errors. A valid question is: what about the other minimum-weight matching, which connects the two detection events on the right instead of on the left? If the algorithm finds this matching, the detected errors would be on the right side of the white square instead of on the left side (where the red stars are now). However, these two cases are logically the same because they are equal up to multiplying with the X-type stabilizer in place of the white square. If we

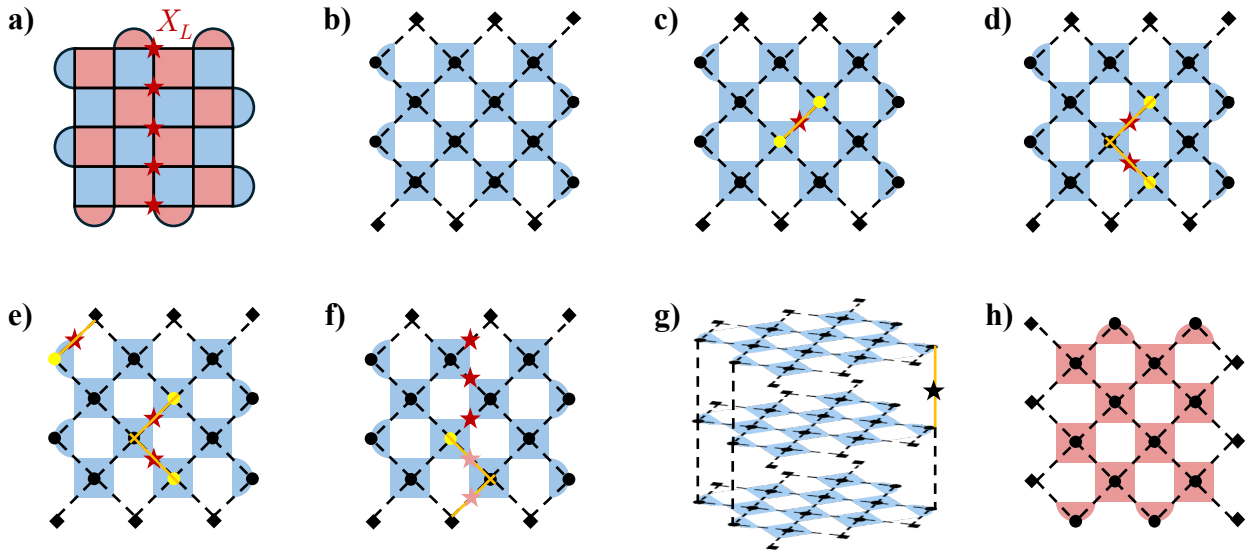


Figure 9.5: Decoding for surface codes. **a)** Surface code layout. A logical X is annotated by red stars. **b)** Decoding graph for Z checks. **c-e)** Examples of successfully decoding errors. Red stars are X errors. Highlighted dots are detection events, i.e., the Z check is flipped. **f)** An example of logical error. The decoding algorithm concludes the X errors represented by the orange stars happened instead of the red stars. **g)** Repeating syndrome extraction to account for measurement errors. For visual clarity, not all vertical edges are drawn. **h)** Decoding graph for X checks.

correct for one case while the other case actually happens, an X-type stabilizer is applied to the qubits in combination, which has no logical effect since it is a stabilizer.

The error correcting capability of surface code is not entirely captured by the distance. If the errors are somewhat 'local', we can correct more errors. For example, in Figure 9.5e, there are three X errors. One of them is desolate from the other two. In this case, the decoding algorithm can correctly identify the error. In comparison, Figure 9.5f also shows three X errors but connected. In this case, only one detection event is there, and the decoding algorithm matches it to the lower boundary. This results in a logical error because the decoded error are the orange stars, in combination with the original error (red stars), we apply a logical X gate (Figure 9.5a) that alters the quantum data.

In general, any error with weight $< d$ can be detected, and any error with weight $\lfloor \frac{d-1}{2} \rfloor$ can be corrected. If we decode an error paulistring P, we can just apply P to the qubits to cancel the error. A surface code with d-by-d data qubits has distance d. In our example, we can detect up to weight-4 errors and correct up to weight-2 errors. The errors in Figure 9.5c and d have weight ≤ 2, which can be detected and corrected. The error in Figure 9.5e has weight 3, but it can still be corrected. The error in Figure 9.5f has weight 3 as well, and it can be detected but not corrected. If we scale up the surface code patch, we can detect and correct higher-weight errors. In each code patch, there are d^2 data qubits, and $d^2 - 1$ ancilla qubits for the syndrome extraction.

One fact we have seen in Section 1.2 is that measurements also have non-perfect fidelity. What if the measurement that produces the Z-type syndromes are wrong? We can account for this type of error by repeating the syndrome extraction circuit many times, and perform graph matching on a 3D graph, as illustrated in Figure 9.5g. The weights of the graph can be adjusted based on the specific error rates of measurements versus the possibility of other types of errors. A measurement error (black star) will set off two detection event connected by a vertical edge that corresponds to this very measurement. Apart from Z checks, we can decode the X checks separately on another graph as shown in Figure 9.5h and combine the results of both to derive the total error.

9.3 Fault-Tolerant Quantum Computing with Surface Codes

Quantum error correction is foundational to fault-tolerant quantum computing, but it is just the beginning. Primarily, QEC preserves quantum data without actively computing; another crucial point is the imperfect syndrome extraction. We construct parity measurements with gates as shown in Figure 1.2c-d, indicating that syndrome extraction itself can introduce additional errors due to the error in these gates. In fact, with low-fidelity gates and suboptimal QEC protocol designs, it is possible that error correction might do more harm than good.

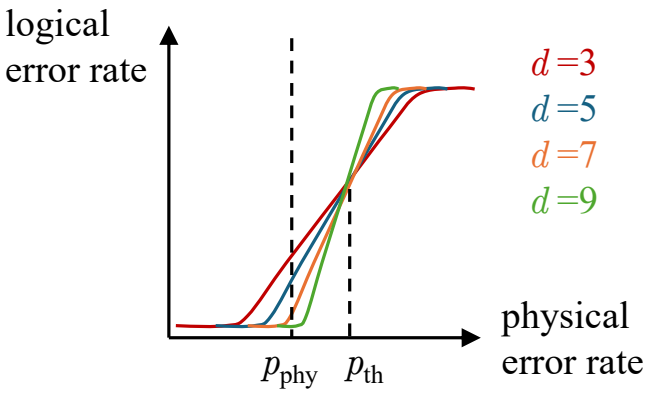


Figure 9.6: Threshold theorem. p_{th} is the threshold. p_{phy} is our physical error rate that is below the threshold. Different code distances (curves with different colors) support different logical error rates given a fixed p_{phy}.

In fault-tolerant quantum computing, errors on every operation are modeled, including gates, initializations, and measurements. A fundamental theoretical result in this field is the threshold theorem [AB08, KLZ98, Kit03], which states roughly that if the physical error rate of all quantum operations is below a certain *threshold*, p_{th}, then QEC can suppress errors to arbitrarily low levels by leveraging more physical qubits. The relationship between logical and physical error rates for the surface code is sketched in Figure 9.6. The specific value of p_{th} depends on the quantum error model, quantum operation error rates, and the decoding algorithm used. Interested readers are referred to [FSG09] for details, which estimates the threshold for surface codes at about 1%.

Despite the error rates mentioned in Section 1.2 being under 1%, a large-scale fault-tolerant quantum computer has yet to be realized. There are several reasons for this. First, for the threshold theorem to be applicable, all quantum operation errors need to be below p_{th}, whereas the rates we cited are typical values; some parts of the computer might perform worse than these numbers suggest. Secondly, to perform large-scale quantum computations with m logical gates, the logical error rate must be much less than $1/m$ [FMM12]. With a fixed physical error rate $p_{\text{phy}} < p_{\text{th}}$, increasing the code distance reduces the logical error

rate. However, if p_{phy} is very close to p_{th} and a very low logical error rate is required, an astronomically high code distance is needed, resulting in an impractically large number of physical qubits. Ideally, p_{phy} should be significantly below p_{th}, allowing for a code distance that is small enough so that the QEC overhead is manageable while large enough to support the intended computations.

According to estimations from [BMT22], if $p_{\text{phy}} = 0.1\%$, a surface code of distance 27 is necessary, corresponding to 1457 physical qubits for one logical qubit. Data from Section 1.2 indicate that current hardware technology is approaching this level of quality and scale but has not yet arrived. If $p_{\text{phy}} = 0.01\%$, a distance of 13 is required, equating to 337 physical qubits per logical qubit, a 4x overhead reduction compared to distance 27. In this section, we assume that hardware technology is capable of effectively supporting quantum error correction, and we discuss how to implement fault-tolerant quantum computing atop this error correction framework.

9.3.1 Fault-Tolerant Operations

We present a few fault-tolerant operations of the surface code in Figure 9.7 where we use a distance-3 surface code tile. A tile has four boundaries of either X or Z type, indicated by red and blue solid lines, respectively. Two opposite boundaries are of the same type after the tile has been initialized and before it is measured. The logical X operator is the tensor product of physical X on a string of data qubits connecting two X boundaries as indicated by the red dashes. Similarly, the blue dashes indicate logical Z. Figure 9.7b-c cover the initialization of logical $|0\rangle$ and $|+\rangle$, which are simply initializing all data qubits to $|0\rangle$ and $|+\rangle$, respectively. Figure 9.7d-e cover the measurement of logical X and Z, which are simply measuring all data qubits along the X and Z basis, respectively. The logical measurement results are the products of measurements on the red or blue dashes, after error correction. A key result of the above definition is that single-qubit Pauli gates can be applied *off-chip* (in the classical control system) by interpreting measured data differently, e.g., if there is a

logical Z gate right before a logical X measurement, we can just measure X and flip our result, instead of actually applying the Z gate *on-chip* with a string of physical Z gates.

Figure 9.7f introduces an operation named *domain wall* realized by applying a layer of physical Hadamard gates (gold H) on all data qubits and then two layers of SWAPs (magenta arrows) to shift data qubits. After the Hadamards, the type of stabilizers of the surface code tile changes, as reflected by the changes in face colors and the changes in colors of the X and Z logical operators after step (1). Note that a domain wall does not implement a logical Hadamard gate, which would require us to rotate the tile by 90 degrees after step (1). This rotation is highly nontrivial since we are assuming fixed physical qubits with nearest-neighbor connectivity. A rotation protocol taking $3d$ QEC rounds in time (where d is the code distance) and an extra ancilla tile in space is presented in [Lit19a].

Here, instead of rotating the tile, we shift it with steps (2) and (3) with SWAPs between data qubits and ancillas such that the types of stabilizers align with the initial configuration. For example, the bottom left four data qubits support a four-body Z check before step (1); now there is still a Z check, albeit two-body. After the domain wall, the boundaries of the tile are rotated by 90 degrees, yet the directions of the logical operators stay the same. Logical Z used to be the Z product of qubits on the blue horizontal dashes, it is still horizontal after step (4), but it becomes the X product of the qubits on the red dashes. The domain wall does not correspond to any single-qubit gate on the logical level, but it is useful when we need to switch the tile boundaries to a desirable orientation to interact with other patches.

Figure 9.7g stands for initializations or measurements along the Y basis following [Gid24]. The specific protocol to realize this operation consists of $d/2$ layers of physical gates. Here we treat such protocol as an atomic operation and do not dive into the details. Since measuring in the Y basis is the same as applying an $S^\dagger$ and measuring in the X basis, and initializing in the Y basis is same as $S|\pm\rangle$, these Y basis operations enable logical S gates.

Lattice surgery operations consist of merge and split on X or Z boundaries [BH20, HFD12]. When tiles merge, they can become non-square patches. In this dissertation, we

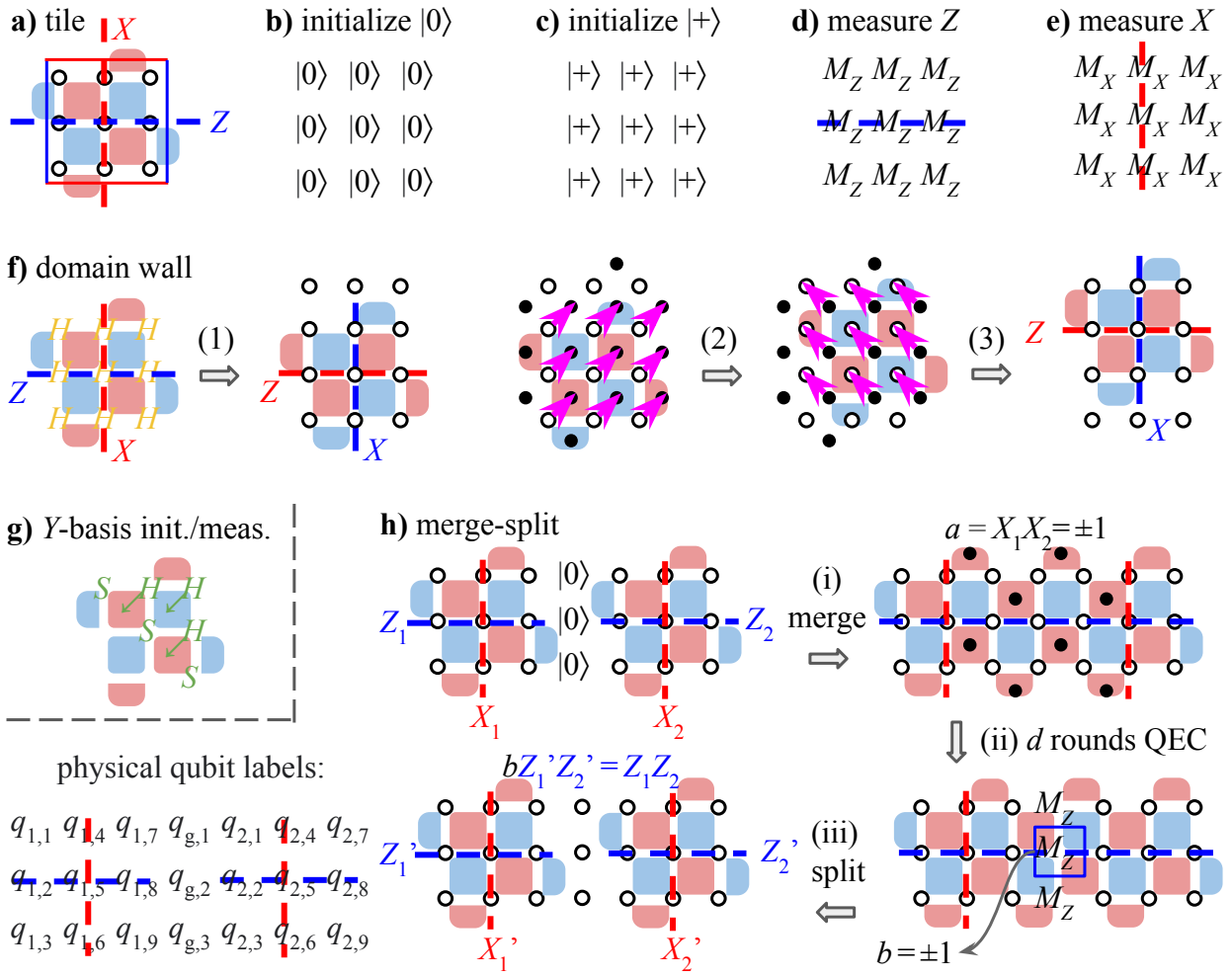


Figure 9.7: Surface code operations. **a)** Surface code tile (simplest patch). **b-e)** Single-patch initialization and measurement in the X and Z basis. **f)** Domain wall. In step (1), Hadamards (gold H) are applied to all data qubits switching the type of stabilizers in the surface code, i.e., color of faces. To realign these colors with other tiles, steps (2) and (3) apply two layers of SWAPs between data and syndrome qubits(magenta arrows) to shift the tile by one unit. **g)** Sketch of Y-basis initialization and measurement [Gid24]. These take half of the code distance many QEC rounds. **h)** Multiple-patch operation with lattice surgery. We refer to the physical qubits with the labels on the bottom left corner. The merge-split implements an XX measurement on two logical tiles.

employ a composite operation *merge-split*: merge patches, perform d rounds of QEC, and then split patches [FG19]. This merge-split definition is slightly restricted in the sense that a split does not have to be *immediate* after the d rounds of QEC. However, we believe that keeping the merged patch does not offer additional benefits. So, we follow this merge-split definition from [FG19].

Figure 9.7h presents the simplest merge-split on Z boundaries of two patches where the labels for involved physical qubits are provided in the left bottom corner. In the merge, we initialize the data qubits in the "gap" between two patches to $|0\rangle$ ($q_{g,1}$, $q_{g,2}$, and $q_{g,3}$) and then adjust the syndrome measurements around this gap as if it is internal of a tile. Concretely, we adjust two Z checks and add four X checks. Before the merge, we have a two-body Z check on $q_{1,8}$ and $q_{1,9}$. During the merge, we perform a four-body Z check on $q_{1,8}$, $q_{1,9}$, $q_{g,2}$, and $q_{g,3}$ instead. Similarly, there was a two-body Z check on $q_{2,1}$ and $q_{2,2}$; during the merge, we perform four-body Z check on $q_{2,1}$, $q_{2,2}$, $q_{g,1}$, and $q_{g,2}$ instead. The new X checks are: four-body on $q_{1,7}$, $q_{1,8}$, $q_{g,1}$, and $q_{g,2}$; four-body on $q_{2,2}$, $q_{2,3}$, $q_{g,2}$, and $q_{g,3}$; two-body on $q_{1,9}$ and $q_{g,3}$; and two-body on $q_{2,1}$ and $q_{g,1}$.

Logically, the merge implements an X_1X_2 measurement. The measurement result, a, is the product of the syndrome qubit measurements indicated by the solid dots after step (i) in Figure 9.7h, including a two-body X syndrome on $q_{1,4}$ and $q_{1,7}$, a four-body X syndrome on $q_{1,7}$, $q_{1,8}$, $q_{g,1}$, and $q_{g,2}$, and so on. Multiplying all these together, the result is exactly the product of X_1 and X_2: visually, this corresponds to the fact that the red faces with the black dots connect the two red dashed lines. After the merge, the new Z operator (long blue dashes) is the product of Z_1 and Z_2.

To reliably readout the value of a, d QEC rounds are appended after the merge. Afterwards, we perform the split where the data qubits in the gap are measured in the Z basis. Suppose the measurement in the box (on $q_{g,2}$) after step (ii) in Figure 9.7h yields result b, we have $bZ_1'Z_2' = Z_1Z_2$. This measurement does not touch X operators, so $X_1'X_2' = X_1X_2 = a$. In summary, a merge-split of two tiles on Z boundaries implements a logical XX measure-

ment; similarly, a merge-split on X boundaries implements a ZZ measurement. We need to record the measured values a and b to determine possible off-chip Pauli corrections as we will discuss in Section 9.3.2.3.

9.3.2 Fault-Tolerant Logical Block

In the previous subsection, we have introduced protocols that can perform S, H, and Pauli gates. The CNOT gate, as decomposed in Figure 1.2e, can be implemented through two lattice surgery merge-splits along with some simpler operations, as depicted in Figure 9.8a. Therefore, we are equipped to perform any Clifford circuit fault-tolerantly since H, S, and CNOT are universal for Clifford circuits [NC10, Got97]. However, to execute any general quantum circuit, we still require a non-Clifford gate, which will be discussed subsequently.

In this subsection, we explore extending the definition of Clifford circuits to include non-unitary operations. This extension is natural as some fundamental operations like lattice surgery merge-splits are inherently non-unitary. It is common in quantum computing for a logical block [BDM23] of computing to have a differing number of inputs versus outputs. For instance, certain protocols generating specific states have only outputs but no inputs. In [TNG24], which is detailed further in Chapter 10, such extended Clifford operations, composed from the atomic operations listed in Figure 9.7, are referred to as lattice-surgery subroutines (LaS).

9.3.2.1 Pipe Diagram Representation for Lattice-Surgery Subroutine

To reason about a logical block, we do not need all the information in Figure 9.7. For instance, the code distance, d, is decided based on fidelity of physical gates and the total error budget and is independent from the computation carried out in a LaS. To represent the on-chip process in a distance-independent manner, we can use time slices such as Figure 9.8a. These operations implement a CNOT between the control tile (C) and the target tile (T)

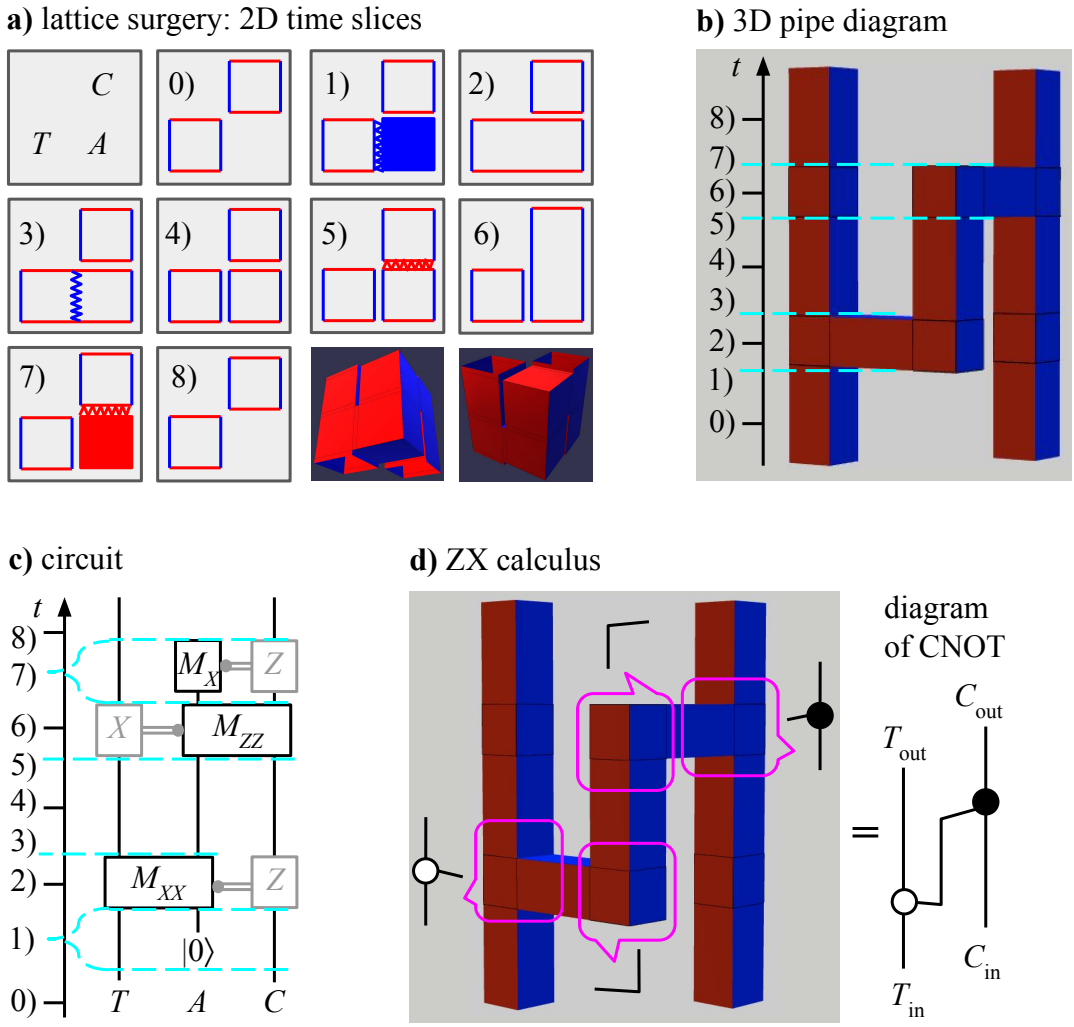


Figure 9.8: Logical CNOT in different representations. **a)** 2D time slices for lattice surgery. The floor plan is in the upper-left corner (C: control, T: target, A: ancilla). 1-3) show the first merge-split. 5-7) show the second merge-split. 0), 4), and 8) are the states between these operations. **b)** 3D pipe diagram for lattice surgery. We mark the time slices in a) on the time axis. **c)** CNOT quantum circuit using parity measurements. We also mark the time slice in a) on the time axis. Slice 1) includes the initialization. Slice 7) includes the X measurement. The feed-forward gates (gray) are Pauli which do not need to be performed on the quantum chip. **d)** ZX calculus. Junctions of pipes correspond to ZX spiders. The whole pipe diagram maps to the ZX diagram of a CNOT.

via an ancillary tile (A). Each tile corresponds to $d \times d$ data qubits physically, but we only need to keep its boundaries in the representation to reason about the logical effects of operations. Slices 1-4) correspond exactly to a merge-split on Z boundaries as exhibited in Figure 9.7h, while slices 4-7) correspond to a merge-split on X boundaries. The color-filled tiles correspond to logical initializations or measurements as Figure 9.7b-e.

Why does this sequence of operations implement a CNOT? We can compare the quantum circuits of a CNOT using parity measurements [FG19], Figure 9.8c, with the time slices. Slice 1) simultaneously initializes A to $|0\rangle$ and starts a merge-split on Z boundaries of A and T, i.e., the XX measurement of A and T. Slice 2) is during the d rounds of QEC in the merge-split, and slice 3) is finishing the merge-split. Note that the gray components in the circuit are Pauli gates depending on the measurement results. They do not appear in the slices because they are applied in the classical control software of the quantum chip. Slices 4-7) correspond to the second merge-split between A and C and also the X measurement of A.

The time slices are "snapshots" of what happens on the quantum chip at important moments. Many moments during QEC are neglected because they would be the same with previous moments. To also capture the time dimension in the representation, we introduce the 3D pipe diagram. A tile of distance d needs d rounds of QEC after an operation, so its boundaries "stay put" for d rounds. When we trace these boundaries on a time axis, a tile accumulates to a cube. The two small 3D drawings in Figure 9.8a exhibit such *in-scale* pipe diagrams for the CNOT. However, the merge-splits in the in-scale drawings are in the narrow gaps between cubes, not very visible. Therefore, we elongate the distance between the cubes so that the merge-splits become "pipes" as shown in Figure 9.8b. The 3D pipe diagram is simpler and more intuitive than the slices, e.g., a merge-split is just a horizontal pipe instead of multiple slices. The continuations and terminations of vertical pipes indicate initializations and measurements. For example, the middle pipe terminates at 7) when the ancilla is measured out in the X basis.

If a pipe is open-ended, it is called a *port*. In Figure 9.8b, there are four ports: the inputs

of C and T that feed in the two tiles from below, and the outputs of C and T that exit the subroutine on the top.

9.3.2.2 Lattice-Surgery Subroutine and ZX Calculus

It turns out lattice surgery has a very natural correspondence to ZX diagrams [BH20, BPW17, BLN24]. As a result, we can derive the ZX diagram of a pipe diagram in a straightforward way (Figure 9.8d) [GF19b]: every cube is a spider, and every pipe is a wire connecting two spiders. If a cube is simply a 90-degree turn, it is a wire. For a T-junction or a cross-junction, the color of the junction is the color that draws the T or the cross. In Figure 9.8d, there is a red T-junction on the lower left and a blue T-junction on the upper right. Then, the spiders have corresponding types, e.g., the red T-junction is an X-spider whereas the blue T-junction is a Z-spider. All the phases of the spiders are 0 except for Y-basis initialization and measurements which have phase $\pi/2$. Wiring these spiders together in Figure 9.8d, we find that the pipe diagram indeed implements the ZX diagram of a CNOT (Figure 9.2c).

A more complex example, majority gate, is presented in Figure 9.11. From the pipe diagram in Figure 9.11d, we extract a ZX diagram located on the top left of Figure 9.11e. The vertical pipes with golden rings are domain walls (Figure 9.7) which maps to a 'Hadamard edge' in ZX calculus [GF19b]. In Figure 9.11e, we use ZX rewrite rules to prove that the extracted diagram is equivalent to the ZX diagram of a majority gate, Figure 9.11b (provided by [GF19b]). In the first step, we apply the Hadamard inversion rule (orange) in the callout and the spider-merge rule (magenta) in Figure 9.2c. In the second step, we morph the diagram without altering connectivity (numbers annotate corresponding spiders). Next, we apply the (generalized) Hopf rule (green), illustrated in Figure 9.11c: when Z- and X-spiders are connected in a complete bipartite manner, two new spiders can replace them. Applying this rule twice reconstructs the ZX diagram in Figure 9.11b, completing our proof.

It is certainly valuable to use ZX calculus to confirm the correctness of the pipe diagram.

However, since ZX calculus does not have the notion of tile orientation, it is challenging to directly synthesis the LaS with ZX calculus.

9.3.2.3 Lattice-Surgery Subroutine Functionality and Correlation Surfaces

We have just proved that Figure 9.8b implements a CNOT by establishing equivalence with the circuit in Figure 9.8c. However, it is hard to construct LaS entirely from circuits because 1) lattice surgery is non-unitary, different from the unitary gates we are used to; and 2) some critical information are missing, like the layout and orientation of tiles which determine whether a merge-split is valid. We can ensure the correctness of LaS with objects called *correlation surfaces* [RHG07, Pal15] that travel inside the pipes.

In Figure 9.9a, we exhibit a correlation surface ensuring the LaS satisfies a stabilizer flow $Z_T \to Z'_C Z'_T$, one of the flows that define a CNOT. The correlation surfaces for the other three flows should also be identified to ensure the correctness of this LaS. These seemingly complicated surfaces are composed from simple local pieces at each junction of pipes, e.g., Figure 9.9a is composed by Figure 9.9c and g (by stitching over Z_A).

A correlation surface relates a set of quantum logical operators and a set of measurement results. The operators are at all the ports this surface propagates to, and the measurements are the projections of this surface on the ceiling of all pipes it propagates in. In Figure 9.9c, the operators are Z_T, Z'_T, and Z_A (highlighted in cyan), and the measurements are on the highlighted yellow line. 'Correlation' just means the product of these logical operators are equal to the product of the measurements. So, why is this local piece valid? Recall that the horizontal pipe in Figure 9.9c is a merge-split on Z boundaries as in Figure 9.7h. Since the ancilla is initialized to $|0\rangle$, its Z value before the merge-split is fixed, $\langle 0|Z|0\rangle = 1$. Thus, according to Figure 9.7h, $bZ'_T Z_A = Z_T \cdot 1$, i.e., $Z'_T Z_A Z_T = b$. This is consistent with our definition since the left hand side is the product of all logical operators the surface touches, and the right hand side is the measurement on the projection of the surface to the ceiling of the horizontal pipe. Note that we have elongated the horizontal pipes in the 3D

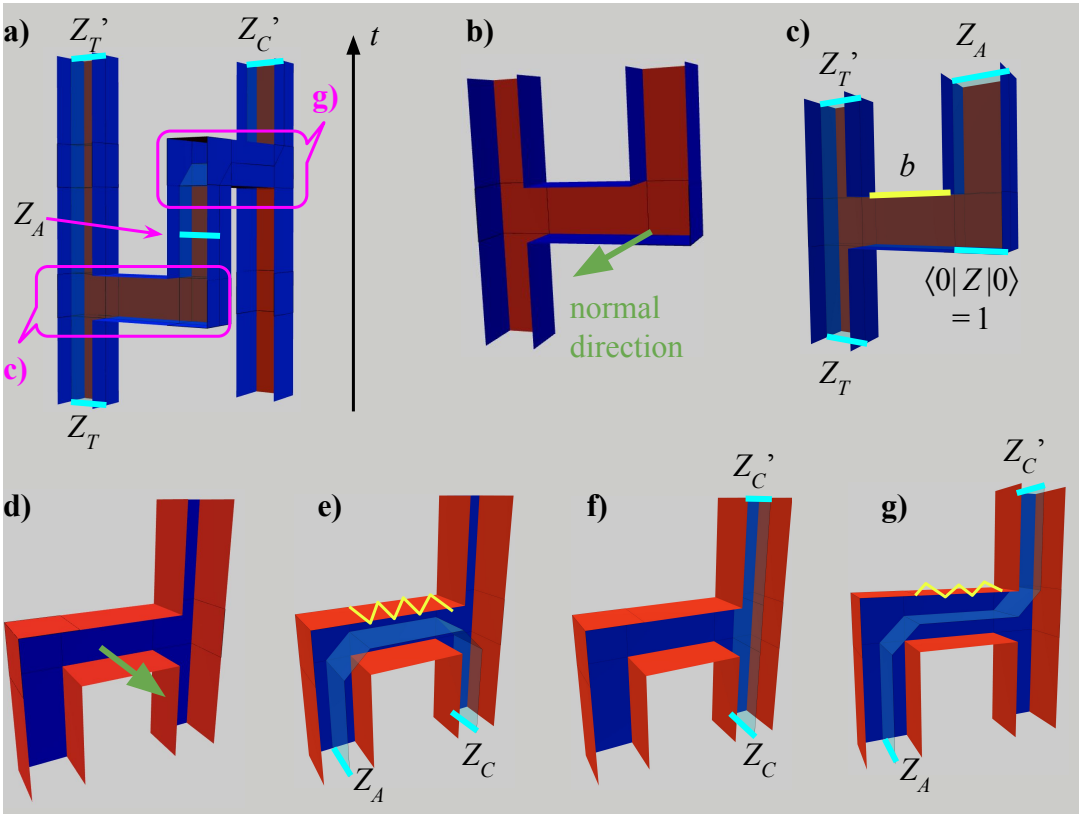


Figure 9.9: Examples of correlation surfaces. **a)** A surface that ensures the stabilizer flow $Z_T \to Z_T' Z_C'$ in a CNOT pipe diagram (front face removed). It is a product of two local pieces in c) and g). **b-c)** Two cases where the surface is orthogonal to the normal direction. **d-g)** Four cases where the surface is parallel to the normal direction. The sign of the correlated operators is the product of measurements on either a line of qubits, highlighted in c), or a rectangular region of qubits, indicated by the wiggled lines in e) and g).

diagram. The highlighted line corresponds to only one measurement physically, which is the one producing b in Figure 9.7h. In real experiments, we will need the correlation surfaces to tell us which set of measurements decides the sign of these product of operators. If we require $Z_T Z'_T Z_A = 1$ while b measures -1, some logical single-qubit Pauli gates should be applied off-chip to fix it.

No surface at all, e.g., Figure 9.9b, is also valid. In general, the *rules* of correlation surfaces at junctions can be reasoned from the logical effects of merge-splits. There are two categories: whether the surface is orthogonal to the *normal direction* of the junction, or parallel to it. The normal direction is orthogonal to all the pipes in the junction, as indicated by the green arrows in Figure 9.9b and d. If the surface is orthogonal to it, the surface correlates all (Figure 9.9c) or none (Figure 9.9b) of the logical operators at ports. If the surface is parallel to it, the surface correlates an even number of logical operators at ports: in our example, there can only be 0 (Figure 9.9d) or 2 operators (Figure 9.9e-g). The wiggly lines in Figure 9.9e and g indicate the projections of correlation surfaces to the ceilings of pipes, which correspond to the parity measurement result a in Figure 9.7h. (Although this particular horizontal pipe is a ZZ measurement instead of an XX measurement.) We will formally present all the rules for correlation surfaces in Section 10.1.4. If at every junction, these rules are respected, the whole correlation surface is valid.

9.3.2.4 Examples of Lattice-Surgery Subroutine

In Figure 9.10b, we present a subroutine which generates a graph state. The stabilizers of a graph state with underlying graph $G = (V, E)$ are generated by $\{X_i \cdot \prod_j Z_j \mid \forall i \in V,\ (i, j) \in E\}$, i.e., X on a node and Z on all its neighbors in G, e.g., Figure 9.10a. These states have many applications in FTQC [HDE06, VPG24]. It should be noted that graph state generation is a non-unitary subroutine since there are no inputs and only outputs.

We provide another example, the majority gate, in Figure 9.11d. The functionality of a LaS is specified by stabilizer flows, implying it can always be implemented using Clifford

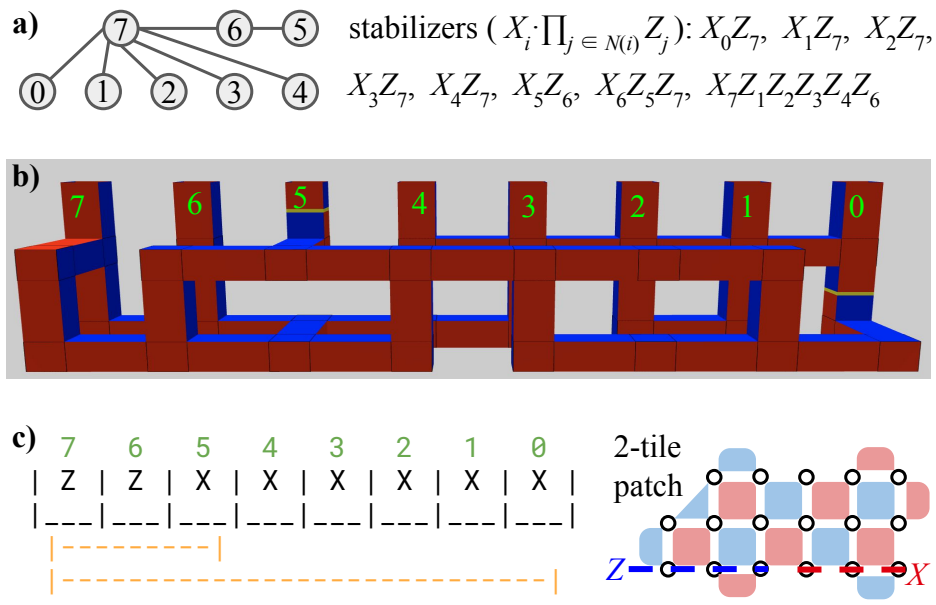


Figure 9.10: Graph state. **a)** Stabilizer generators of graph states. **b)** 8×2×2 LaS generating the graph state in a). **c)** 8×4×2 solution found by [LBM23]. They require 2-tile patches instead of 1-tile patches in our case. Letters indicate the initial bases of patches. Orange intervals indicate two layers of parity measurements.

gates and Pauli measurements. However, the majority gate is non-Clifford, so lattice surgery alone cannot fully implement it. Typically, non-Clifford resources are generated in dedicated regions on the quantum chip and routed to specific places. By considering non-Clifford resources as input to certain ports, the remaining portion constitutes a LaS specified by stabilizers. Thus, our LaS definition is enough for general FTQC subroutines. Additionally, this definition is not limited to specific types of non-Cliffordness; for example, this majority gate [GF19b] consumes a $|\mathrm{CCZ}\rangle$ instead of $|T\rangle$.

9.3.3 Distillation Factory

We have seen a majority gate must consume non-Cliffordness, but where do those come from? A prevalent solution are magic state distillation factories (usually for $|T\rangle$ or $|\mathrm{CCZ}\rangle$)

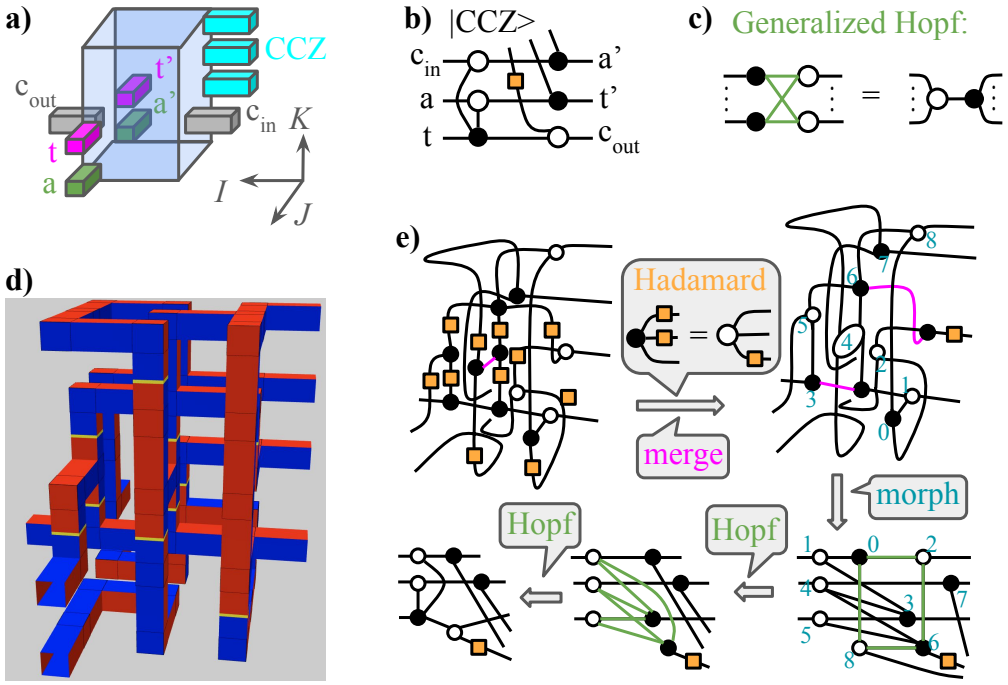


Figure 9.11: Majority gate optimization. **a)** Port requirements. [GF19b] provides a 3×5×5 design. **b)** ZX diagram of the gate. **c)** Hopf rule in ZX calculus. **d)** A 3×3×5 design. **e)** ZX calculus proof of the LaS.

that consume noisy magic states and produce higher-quality magic states. Our LaS model can support such factories: a first-level factory involves physical magic state injections at certain ports, while a higher-level factory takes already distilled magic states to the ports; the remaining part can still be specified by stabilizers. Distilling injected $|T\rangle$s to a usable error rate entails thousands of operations, making non-Clifford gates a significant FTQC cost. Reducing distillation factory sizes directly reduces this dominant cost.

The 15-to-1 T-factory is one of the most realistic choices for early FTQC, visualized in Figure 9.12b in the circuit model. The 15 $T^\dagger$'s and the final output $|T\rangle$ correspond to 16 non-Clifford ports in our LaS. The remaining portion is specified by stabilizers at the locations marked with orange labels, derived in Figure 9.12c. A baseline design, manually optimized by experts in [FG19, GF19a], initializes logical qubits in $|0\rangle$ with unit depth, an

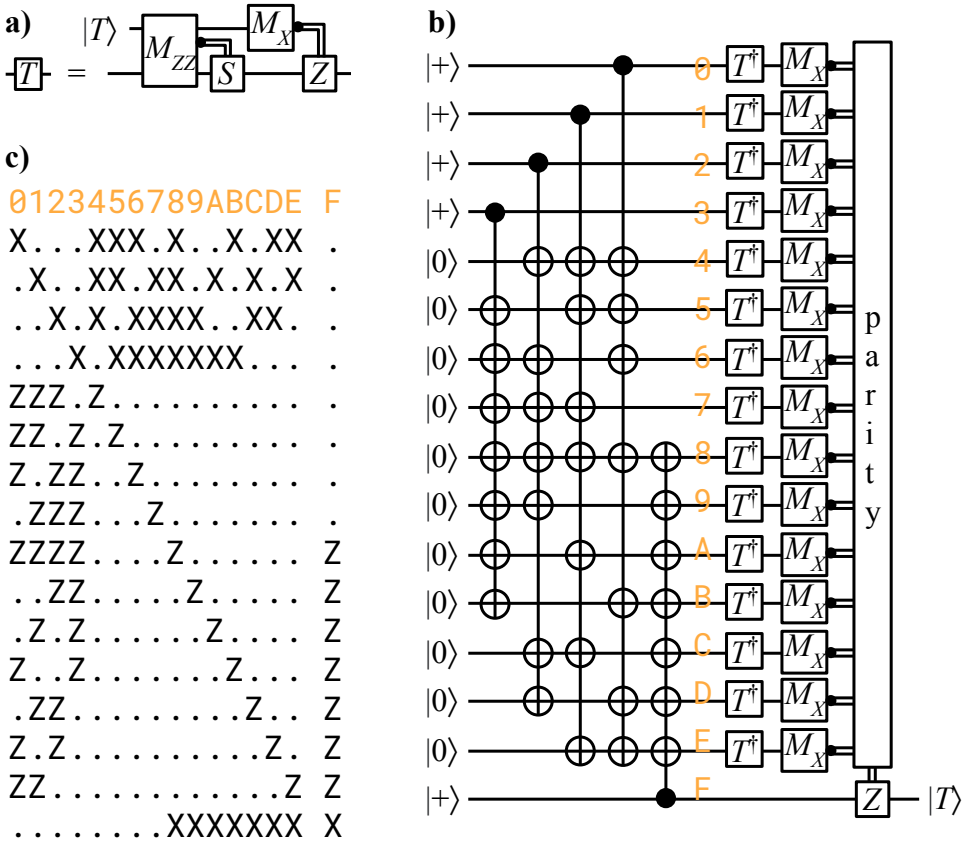


Figure 9.12: Fault-tolerant T. **a)** Implementing a T gate by consuming a magic state. **b)** Quantum circuit for the 15-to-1 T-factory [FG19]. **c)** Stabilizers of the underlying [[15,1,3]] error correcting code.

eigenstate of the Z stabilizers in Figure 9.12c. It then measures the 5 8-body X stabilizers in Figure 9.12c, each taking unit depth. A layer of possible *fixups* consisting of half-distance Y-basis measurement as in Figure 9.7g is appended (see Section 10.3.3 for details), resulting in a total LaS depth of 6.5. When used repeatedly, one unit of latency is hidden through interleaving, and this baseline factory averages a depth of 5.5. With a footprint of 8×4=32, this baseline factory has an average volume of 32×5.5=176.

Distillation factories are assigned to dedicated regions in fault-tolerant architectures. An example architecture based on surface codes is depicted in Figure 9.13, as discussed in

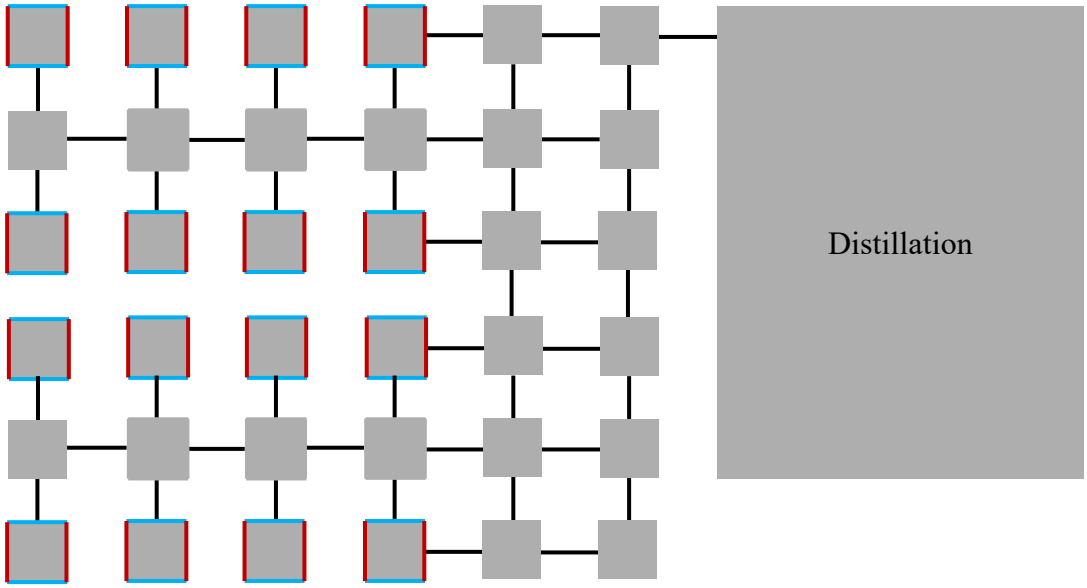


Figure 9.13: An example fault-tolerant architecture layout.

[FG19]. This configuration contrasts with the layout shown in Figure 1.11c by having a fixed orientation for qubits (tiles with boundaries). Each qubit has access to nearby ancillas (those without boundaries) to facilitate the implementation of H or S gates. Thanks to the abundance of ancillas, lattice surgery can be performed between any two qubits, enhancing the architecture's versatility. The distillation factory produces the magic state located at its top left corner, which can perform lattice surgery with any logical qubit via ancilla paths. This enables us to apply fault-tolerant T gates to arbitrary qubits with the protocol in Figure 9.12a. Therefore, the hypothetical architecture in Figure 9.13 supports universal fault-tolerant quantum computing.

CHAPTER 10

LaSsynth: Representation and Synthesis of Subroutines for Fault-Tolerant Quantum Architectures Based on Surface Codes

The toy example depicted in Figure 9.13 is not optimal, as it incorporates some artificial constraints, such as fixed orientations and placements of the surface code tiles. In this chapter, we develop a method to synthesize lattice-surgery subroutines (LaS) without these constraints, leading to the creation of LaSsynth [TNG24]. This tool is designed to explore the possibilities within a given spacetime volume, pushing the boundaries of what can be achieved with the surface code operations. However, it is important to note that not all computation can be encapsulated within a single LaS. Therefore, a higher-level compiler is still necessary to segment the computation into multiple LaS, strategically place these subroutines within the quantum architecture, and route qubits between them. We would like to remind the reader that this chapter builds upon the notations and concepts introduced in Chapter 9. It is recommended to read that chapter first.

In contrast to previous compilation frameworks from quantum algorithms to lattice surgery [BMT22, CC22, FG19, Lit19a, LN22, PF20, WNS23], the focus of this chapter is on hyper-optimizing small and frequently-used LaS. Our tool is well suited for optimizing basic components with 5-20 qubits and 10-100 operations like the MAJ and UMA gates in adder circuits [CDK04], or magic state distillation factories. The intent is that, when assembling larger computations, the hyper-optimized subroutines created by our tool can be used as

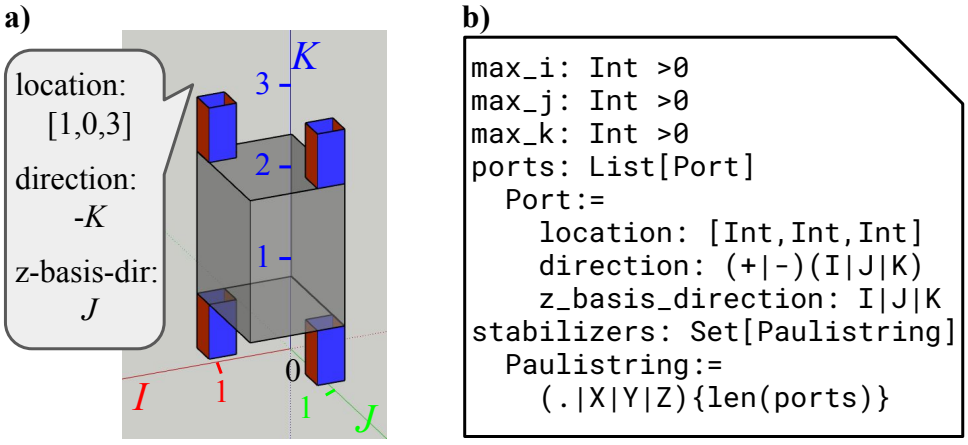


Figure 10.1: Lattice-surgery subroutine (LaS). **a)** Externals of a LaS implementing a CNOT with four ports and $2 \times 2 \times 2$ volume. **b)** LaS specification including volume, port layout, and functionality (stabilizer) information.

building blocks. Another major difference with previous approaches is the removal of human heuristics, aiming for provable optimality. Unlike human constructions guided by some overarching ideas, our tool can explore results that are better but also highly non-intuitive. The output usually comprises intricate arrangements of pipes and junctions, challenging to construct or even comprehend with human intuition.

Figure 10.1 provides the specification of LaS. The underlying surface code substrate is divided into tiles, our unit of space. The tiles are indexed by spatial (I and J axes) coordinates, and they can only interact with nearest neighbors. Similar to a classical circuit component, a LaS takes up a certain area on the quantum chip, i.e., *footprint*. In the example of Figure 10.1a, we allow 2×2 tiles: (0,0), (0,1), (1,0), and (1,1) in the I-J plane. A LaS is a procedure instead of a static entity, so it also has a duration (along K axis). In Figure 10.1a, we allow two units of time: $k = 1$ and 2, each corresponds to a layer of operations and d rounds of QEC afterwards. Aligning with existing literature, we use *spacetime volume*, i.e., the footprint times the duration, as our metric for LaS optimization. The volume is indicated by the half-transparent box in our example. Inside this box are the operations implementing

the function of the LaS. Note that the LaS specification does *not* contain what happens inside this box. Rather, it is the job of a synthesizer or compiler to figure out a solution satisfying the given specification.

How can we specify the function of a LaS? For a classical digital circuit, this can be done by a lookup table between input and output ports. A LaS also has some *ports* connecting the given volume to the outside, e.g., the 4 short pipes in Figure 10.1a. We first need to locate these ports, as in the list of `Port` in Figure 10.1b. The `location` of a port is the outside 3D grid point it connects to, e.g., $(i, j, k) = (1, 0, 3)$ for the called-out port in Figure 10.1a. Then, the `direction` of the port is the direction from this outside point to the inside. In our example, from $(1, 0, 3)$, we need to go down to enter the box, so the direction is $-K$. Finally, because of the two types of boundaries (red/blue), orientation of a port is also important. In our example, the blue (Z) boundary is perpendicular to the J axis, so the value of `z-basis-dir` is J. The four ports in Figure 10.1a are inputs (bottom) and outputs (top) for control (left) and target (right) qubits in a CNOT gate. To express the function of this LaS, the `stabilizers` in Figure 10.1b are *stabilizer flows* on its ports: $ZI \to ZI$, $IZ \to ZZ$, $XI \to XX$, and $IX \to IX$, as introduced in Section 9.1.3. Note that these stabilizer flows are on the logical level, not to be confused with stabilizers measured on the physical level (inside tiles) in QEC rounds.

Our main task is to generate valid LaS given a specification above covering allowed volume, port configurations, and stabilizers to satisfy. The main contributions are identified in Figure 10.2.

1 Representation. Previous FTQC compilation works have introduced instruction sets atop lattice surgery, such as multi-qubit $\pi/8$-rotations and measurements [Lit19a], or a more general 'planar quantum ISA' [BMT22]. A main effort is decomposing arbitrary quantum algorithms into these instruction sets, which then straightforwardly unroll to known lattice surgery implementations. While this abstraction layer of instruction set is necessary for efficiently compiling large-scale algorithms, it falls short when optimizing limited-scale LaS,

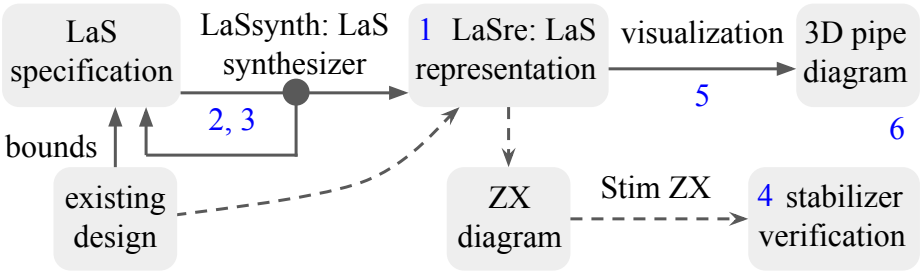


Figure 10.2: Overview of LaSsynth contributions (blue numbers). Solid arrows represent the main workflow. Dashes show the verification workflow for existing designs.

as it cannot explore all potential combinations of lattice surgery and other surface code operations. We present the first LaS representation, LaSre, at the *native* lattice surgery level, such that merging and splitting can potentially happen between any adjacent patches in the plane at any time, thus breaking the previous abstraction layer.

2 Formulation. Some *validity constraints* are required to ensure that LaS expressed in our representation are valid. For instance, two adjacent patches can only merge by the same type of boundary. In the merge illustrated by Figure 1.11b, this constraint means the two pipes need to have the same color of boundaries (blue) facing each other. Beyond validity, we also need to impose some *functionality constraints* such that the resulting LaS realizes the stabilizer flows in the specification. This is done by keeping track of *correlation surfaces* as introduced in Section 9.3.2.3. In summary, our formulation establishes a comprehensive set of constraints for the LaS synthesis problem based on our representation.

3 Synthesizer. Based on our formulation, we built LaSsynth, a software that transforms LaS synthesis into satisfiability (SAT) and queries SAT solvers for solutions. Given a LaS specification like Figure 10.1b, the synthesizer either concludes that it is impossible to implement (UNSAT) or produces a satisfying variable assignment. Existing designs provide us with concrete upper bounds in space and time to implement certain LaS. In order for further optimization, we can start from these bounds, and iteratively generate and solve LaS specifications with even lower bounds, indicated by solid arrows in Figure 10.2. Given the

internal use of SAT by the synthesizer, its primary use case is optimizing frequently used subroutines, which can be pre-derived and integrated with end-to-end FTQC compilers. The evaluations demonstrate that LaSsynth can solve subroutines of practical significance within a reasonable time frame.

4 Verification. We provide a verification workflow utilizing ZX calculus, as illustrated by the dashes in Figure 10.2. Our formulation ensures correctness automatically, so the results generated by LaSsynth do not need to go through this. However, when we were developing this software, the verification was very helpful in debugging. Additionally, we can use it to check LaS designed by others, e.g., we found that the majority gate in [GF19b] does not realize some required stabilizer flows, underscoring the importance of automatic verification.

5 Visualization. Until now, researchers have to manually construct the pipes in professional 3D modeling software [GF19b, FG19]; or reply on 2D time slices like Figure 9.8a [Lit19a, PF20]. These compromises hinder lattice surgery research. In response, we have developed a translation script from our representation to a 3D modeling format, facilitating visualization of LaS as pipe diagrams.

6 Specific designs. We discovered a majority gate design—a frequently used subroutine in Shor's algorithm—with a 40% reduction in volume compared to previous work [GF19b]. We also applied LaSsynth to optimize 15-to-1 T-factories. Leveraging our foundational formulation and exhaustive search, it improves two state-of-the-art designs by 8% than [FG19, GF19a] and by 18% than [Lit19a], under their respective settings.

The rest of this chapter is organized as follows. Section 10.1 presents the formulation of the LaS synthesis problem. In Section 10.2, we provide the details on the implementation of LaSsynth. In Section 10.3, we apply LaSsynth to graph state generation, majority gate, and T-factory. In Section 10.4, we survey previous works.

10.1 Formulation

Given a LaS specification as in Figure 10.1b, we formulate the LaS synthesis problem into binary variables and constraints that must be satisfied. These variables constitute LaSre, our LaS representation. The constraints are of two types: *validity* and *functionality*. The former ensures that the LaSre is a valid FTQC procedure, whereas the latter ensures that it satisfies the stabilizers specified, which is the function of the LaS.

From this point on, we will use I and J as two spatial axes, and K as the time axis. This notational choice is because letters X, Y, Z, and T have other meanings in the context.

10.1.1 Structural Variables

In a pipe diagram, each cube can potentially connect to its nearest neighbors in the 3D spacetime. Thus, to specify the structure of the pipes, all we need is one bit for every adjacent pair of cubes meaning whether there is a pipe or not. These are the `ExistI`, `ExistJ`, and `ExistK` variables below. Additionally, each horizontal pipe has a color orientation (Figure 10.3e), denoted by `ColorI` and `ColorJ` variables below, that corresponds to whether the pipe is a merge-split on Z or X boundaries. Once the configurations of all pipes are known, the faces of cubes can be inferred from all its incident pipes, except for one case–Y cubes, which are represented by green boxes in this chapter. These correspond to initializing and measuring patches along the Y basis (Figure 9.7g) which is necessary to implement logical Clifford operation in general.

There are five arrays of structural variables. Each has shape (`max_i`, `max_j`, `max_k`), i.e., one binary variable per spacetime volume. All indices start from 0. Figure 10.3 shows the values of these variables in the CNOT example.

$\texttt{YCube}_{i,j,k}$ specifies whether the cube at (i, j, k) is a Y cube. In this example, there is none, so all $\texttt{YCube}_{i,j,k} = 0$.

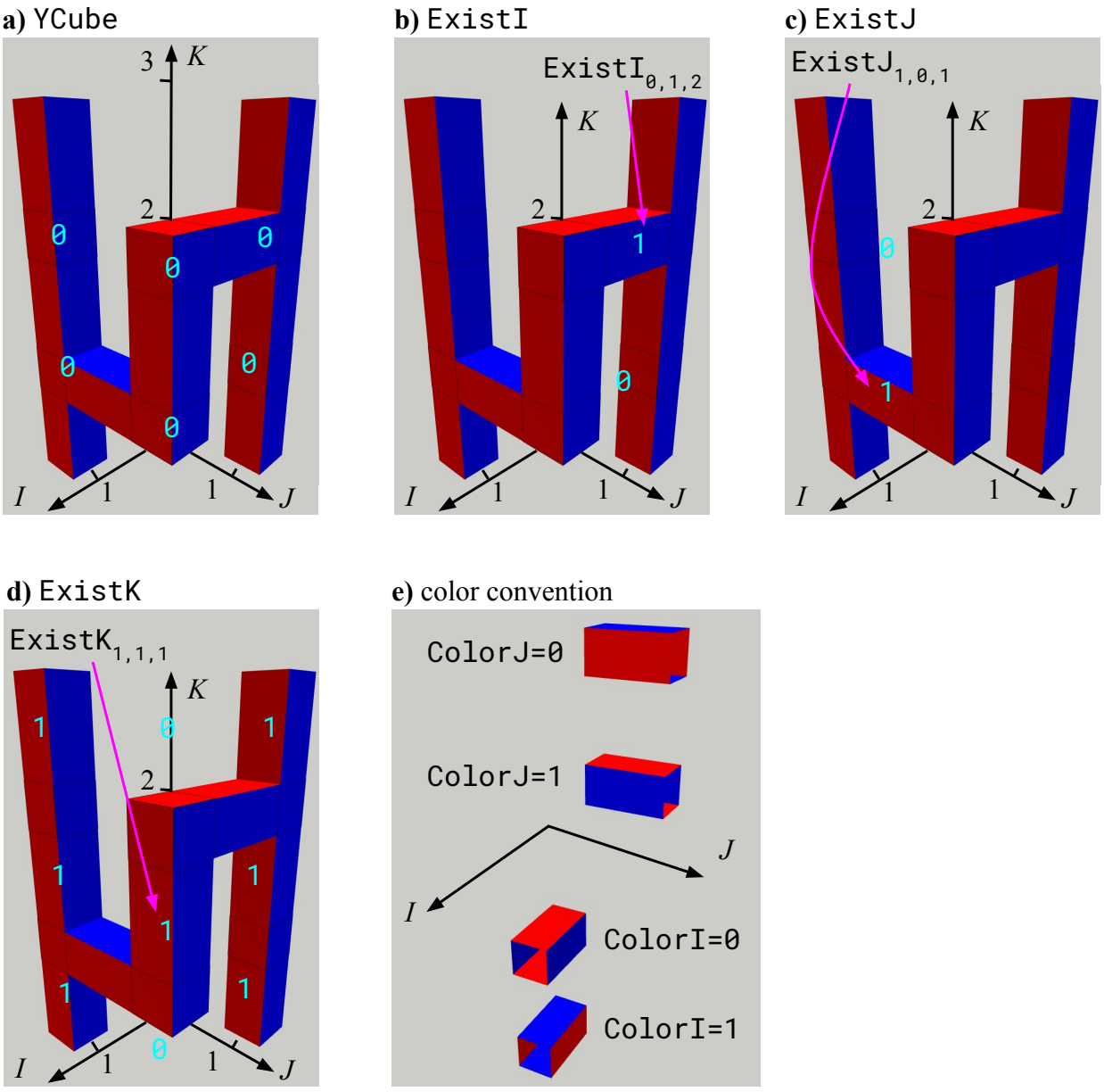


Figure 10.3: Structural variables in LaSsynth. **a)** `YCube` on each cube specifies whether that cube is a Y cube. **b-d)** `ExistI/J/K` on each edge specifies whether there is an edge in the corresponding direction. **e)** color convention for `ColorI/J` variables.

$\texttt{ExistI}_{i,j,k}$ specifies whether there is a pipe from cube (i,j,k) to $(i+1,j,k)$. There is only one pipe in I direction, which is from $(0,1,2)$ to $(1,1,2)$ (see Figure 10.3b). So, only $\texttt{ExistI}_{0,1,2} = 1$, while all other `ExistI` variables evaluate to 0, e.g., directly below the existing I-pipe, $\texttt{ExistI}_{0,1,1} = 0$. Note that our convention allows pipes from $(\texttt{max\_i} - 1, j, k)$ to $(\texttt{max\_i}, j, k)$, which extend out of the confined volume. These can only be ports. However, we do not allow -1 as an index, so there is no pipe from $(-1, j, k)$ to $(0, j, k)$.

$\texttt{ExistJ}_{i,j,k}$ specifies whether there is a pipe from cube (i,j,k) to $(i,j+1,k)$. In the example, there is only one J-pipe from $(1,0,1)$ to $(1,1,1)$, so $\texttt{ExistJ}_{1,0,1} = 1$ (Figure 10.3c), and all other `ExistJ` variables are 0, e.g., $\texttt{ExistJ}_{1,0,2} = 0$.

$\texttt{ExistK}_{i,j,k}$ specifies whether there is a pipe from cube (i,j,k) to $(i,j,k+1)$. There are many K-pipes in the example, indicated by the 1's in Figure 10.3d. Although the merge-splits only concerns horizontal pipes, we still need these variables for vertical pipes because they indicate initializations and measurements. For example, $\texttt{ExistK}_{1,1,0} = 0$, $\texttt{ExistK}_{1,1,1} = 1$, and $\texttt{ExistK}_{1,1,2} = 0$ means the tile at $(i,j) = (1,1)$ is initialized at $k = 1$ and measured at $k = 2$.

$\texttt{ColorI}_{i,j,k}$ specifies the color orientation of I-pipes. Our convention is displayed in Figure 10.3e: if red faces are on the K direction, then $\texttt{ColorI} = 0$; otherwise, $\texttt{ColorI} = 1$. The `ColorI` variables are always used together with `ExistI` variables in the constraints, so if there is no pipe, the corresponding color variable value will not affect the solution.

$\texttt{ColorJ}_{i,j,k}$ specifies the color orientation of J-pipes.

At this point, it seems there should also be `ColorK` variables. Indeed, in the 3D drawings, K-pipes are also colored. However, we do not need `ColorK` variables because the domain wall operation is available to us. This operation is denoted as a yellow ring on a K-pipe, as found in Figure 9.10b (at the green '5'). It can switch the orientation in the middle of a K-pipe to adapt it to any configuration at the two ends. Thus, `ColorK` variables are neglected in the formulation, and inferred in the post-processing based on its neighbors.

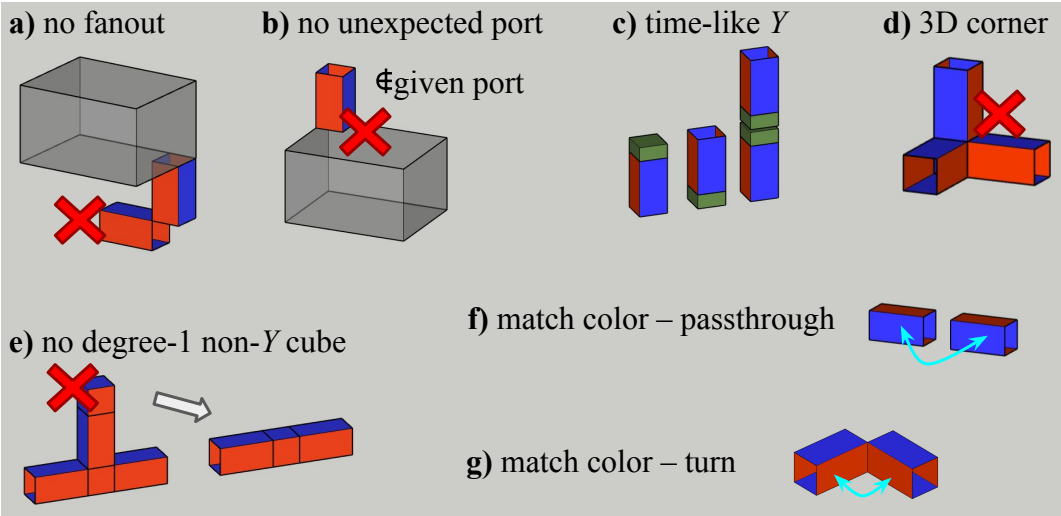


Figure 10.4: Validity constraints in LaSsynth. **a)** Starting point of a port cannot have other pipes. **b)** No pipes other than the ports connect to the outside. **c)** Y cubes can only have K-pipes. **d)** A cube cannot have pipes in all three directions. **e)** All degree-1 non-Y cubes can always "squeeze in", so we forbid such cubes. **f)** Two pipes in a "passthrough" should have the same color orientation. **g)** Two pipes in a "turn" should have matching colors on faces that are touching.

10.1.2 Validity Constraints

Figure 10.4 illustrates the validity constraints. Rules a) and b) follow from our definition of a port; c) is required for the Y-basis operation [Gid24]; d), f), and g) follow from the fact that merge-split can only happen when the adjacent boundaries of two patches have the same type. Rule e) is technically an optimization in order for lower spacetime volume instead of a hard requirement. We provide some examples of these constraints in Figure 10.3 below.

No Fanouts (Figure 10.4a). A port in the LaS specification starts at a cube and connects to *only one* of its six neighbors. For instance, port 0 in Figure 10.5 starts from $(0, 1, 0)$ and connects $(0, 1, 1)$, so there is no connection in the other 5 directions:

$$\overline{\texttt{ExistI}_{0,1,0}}\ ,\quad \overline{\texttt{ExistJ}_{0,0,0}}\ ,\quad \overline{\texttt{ExistJ}_{0,1,0}}\ , \tag{10.1}$$

where an overline means the variable should be 0. Note that the pipes in $-I$ and $-K$ directions for this cube are out of range, so they are automatically neglected.

No Unexpected Ports (Figure 10.4b). Other than the specified ports, there cannot be "dangling" pipes. In Figure 10.5, on the top floor, there are only port 2 starting at $(0, 1, 2)$ and port 3 at $(1, 0, 2)$, so the other two patches cannot connect upwards:

$$\overline{\texttt{ExistK}_{0,0,2}}\ ,\quad \overline{\texttt{ExistK}_{1,1,2}}\ . \tag{10.2}$$

Time-Like Y Cubes (Figure 10.4c). Only time-like (K) pipes are allowed to connect to a Y cube, so

$$\forall(i,j,k)\ \ \texttt{YCube}_{i,j,k}\ \Rightarrow\ \overline{\texttt{ExistI}_{i-1,j,k}}\ . \tag{10.3}$$

We also apply similar constraints with $\overline{\texttt{ExistI}_{i,j,k}}$, $\overline{\texttt{ExistJ}_{i,j-1,k}}$, and $\overline{\texttt{ExistJ}_{i,j,k}}$ on the right hand side.

No Degree-1 Non-Y Cubes (Figure 10.4e). Degree-1 cubes (those having only one pipe) that are neither Y-cubes nor ports can always be "squeezed in". In Figure 10.4e, the degree-1 cube measures logical Z of the patch, but that measurement can be on the horizontal "passthrough" without "popping out". This constraint is expressed as follows: for an endpoint of a pipe, if it is not a Y cube, then at least one of the other five pipes of this cube is present, e.g., for K-pipe $(1, 0, 1)$ to $(1, 0, 2)$,

$$\overline{\texttt{YCube}_{1,0,1}} \wedge \texttt{ExistK}_{1,0,1}\ \Rightarrow\ \big[\texttt{ExistK}_{1,0,0} \vee \texttt{ExistI}_{1,0,1} \vee \texttt{ExistI}_{0,0,1} \vee \texttt{ExistJ}_{1,0,1}\big]. \tag{10.4}$$

The right hand side has 4 terms instead of 5 terms because the pipe in $-J$ direction for cube $(1, 0, 1)$ is out of range.

Matching Colors at Passthroughs (Figure 10.4f). Two pipes in the same direction connecting to a cube should have the same color orientation, e.g., the two possible I pipes of $(1, 1, 2)$:

$$\big[\texttt{ExistI}_{0,1,2}\ \wedge\ \texttt{ExistI}_{1,1,2}\big] \Rightarrow \big[\texttt{ColorI}_{0,1,2} = \texttt{ColorI}_{1,1,2}\big], \tag{10.5}$$

which is satisfied (trivially) because $\texttt{ExistI}_{1,1,2} = 0$.

Matching Colors at Turns (Figure 10.4g). When two pipes in orthogonal directions are touching, their faces that are touching should have the same color. Consider two possible pipes connecting to $(1, 1, 2)$ in $-I$ and $-J$ directions:

$$\left[\texttt{ExistI}_{0,1,2} \ \wedge \ \texttt{ExistJ}_{1,0,2}\right] \Rightarrow \left[\texttt{ColorI}_{0,1,2} \neq \texttt{ColorJ}_{1,0,2}\right], \tag{10.6}$$

which is satisfied because $\texttt{ExistJ}_{1,0,2} = 0$. However, suppose there is a pipe, we can check Figure 10.3e that if the `ColorI` is 0, to match the colors, the `ColorJ` must be 1. Alternatively, the former is 1 and the latter is 0. These two cases are exactly captured by the right hand side with the Boolean $\neq$ operator.

No 3D Corners (Figure 10.4d). If a cube connects to at least one pipe in all I, J, and K directions, it is a "3D corner", which is forbidden. In Figure 10.4d, the K-pipe and the J-pipe have a color matching conflict. Switching the orientation of the K-pipe, then it has a conflict with the I-pipe. This set of constraint is formulated as each cube having a normal direction, i.e., it does not have pipes completely in at least one of the I, J, or K direction. Consider cube $(1, 1, 2)$ again:

$$\left[\,\overline{\texttt{ExistI}_{0,1,2}} \wedge \overline{\texttt{ExistI}_{1,1,2}}\,\right] \vee \left[\,\overline{\texttt{ExistJ}_{1,0,2}} \wedge \overline{\texttt{ExistJ}_{1,1,2}}\,\right] \vee \left[\,\overline{\texttt{ExistK}_{1,1,1}} \wedge \overline{\texttt{ExistK}_{1,1,2}}\,\right], \tag{10.7}$$

which is satisfied since $\texttt{ExistJ}_{1,0,2} = \texttt{ExistJ}_{1,1,2} = 0$.

10.1.3 Correlation Surface Variables

For each stabilizer of a LaS, we need to provide the corresponding correlation surface which can be specified by its pieces inside each pipe. Two pieces are possible inside a pipe: one that connects the two blue faces and one that connects the two red faces. Thus, we need two bits per pipe for each of the correlation surfaces. These are the `CorrIJ`, `CorrIK`, `CorrJK`, `CorrJI`, `CorrKI`, and `CorrKJ` variables below. There are six arrays of correlation surface variables, each has shape (n_{stab}, `max_i`, `max_j`, `max_k`), one binary variable per stabilizer per volume. Figure 10.5 provides the correlation surface variable values in each pipe for the surface seen in Figure 9.9a.

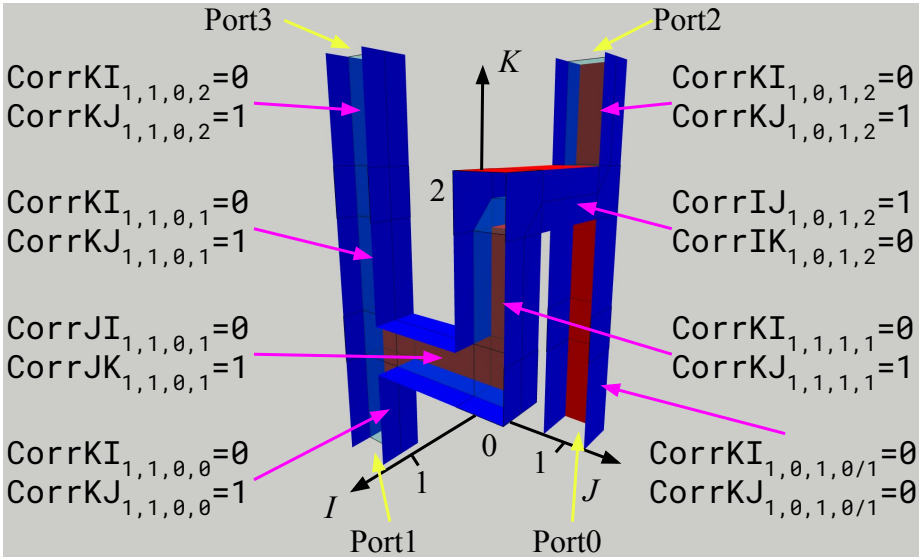


Figure 10.5: Correlation surface variables in LaSsynth. The surface for $IZZZ$ is shown.

$\texttt{CorrIJ}_{s,i,j,k}$ specifies the existence of correlation surface pieces in the I-J plane and inside I-pipes where s is the index of the stabilizer and (i, j, k) is the location of the I pipe. Note that $IZZZ$ has index $s = 1$ among the 4 stabilizers, and there is only one I-pipe from $(0, 1, 2)$ to $(1, 1, 2)$. Thus, $\texttt{CorrIJ}_{1,0,1,2} = 1$.

$\texttt{CorrIK}_{s,i,j,k}$ is for correlation surface pieces in the I-K plane and inside I-pipes. Since there is only one I-pipe, the only nontrivial example here is $\texttt{CorrIK}_{1,0,1,2} = 0$.

Similarly, we have variable arrays $\texttt{CorrJK}_{s,i,j,k}$ and $\texttt{CorrJI}_{s,i,j,k}$ for correlation surface pieces in J-pipes, and $\texttt{CorrKI}_{s,i,j,k}$ and $\texttt{CorrKJ}_{s,i,j,k}$ for those in K-pipes. Since each stabilizer needs a different correlation surface, the number of correlation surface variables scales in the product of the volume and the number of stabilizers, much larger than the number of structural variables scaling only in the volume.

10.1.4 Functionality Constraints

Figure 10.6 illustrates the constraints for correlation surfaces: two at boundaries (a and d), and two at junctions (b and c).

Stabilizer as Boundary Conditions (Figure 10.6a). A stabilizer is specified as a paulistring at the ports, e.g., $IZZZ$ for the four ports of the CNOT. The Z operator touches the two Z boundaries (blue) of the K-pipes of the ports. Given that the `z-basis-dir` in the input is J for all the ports, we find that there should be correlation surface pieces in the J-K plane inside the K-pipes of ports 1, 2, and 3. Also, the correlation surface pieces corresponding to X operators, in the K-I plane, should not be present at these ports. Therefore,

$$\texttt{CorrKJ}_{1,1,0,0},\ \texttt{CorrKJ}_{1,0,1,2},\ \texttt{CorrKJ}_{1,1,0,2},\ \overline{\texttt{CorrKI}_{1,1,0,0}},\ \overline{\texttt{CorrKI}_{1,0,1,2}},\ \overline{\texttt{CorrKI}_{1,1,0,2}}. \quad (10.8)$$

Port 0 should not have correlation surface pieces in either direction since in the stabilizer, its term is I. If the term for a port is Y, then it should have both correlation surface pieces.

Both or None at Y Cubes (Figure 10.6d). Since the Y operator is the product of Z and X, it correlates the two operators. Thus, two correlation surfaces can end at a Y cube together, or neither of them are present in this region. Per Figure 10.4c, only K-pipes can connect to Y cubes, so this set of constraints is

$$\texttt{YCube}_{s,i,j,k} \ \Rightarrow\ \left[\texttt{CorrKI}_{s,i,j,k} = \texttt{CorrKJ}_{s,i,j,k}\right], \quad (10.9)$$

for all tuples of (s, i, j, k).

A non-Y and non-port cube can only have degree 2, 3, or 4: degree-1 cube is forbidden by Figure 10.4e; and degree-5 or -6 cubes will always contain a 3D corner which is forbidden by Figure 10.4d. Degree-2 cubes are "identity" where correlation surfaces trivially travel through. We can neglect them for now and check later that this case is consistent with the two sets of constraints to introduce below. In conclusion, we are only left with degree-3 or degree-4 cubes which can only be T-junctions or cross-junctions, and they each has a normal direction orthogonal to the plane that draws the T or the cross. In Figure 10.6b-c, the T-junctions are in I-J plane, so their normal direction is K. The two constraints below depend on whether the correlation surfaces are orthogonal or parallel to the normal direction.

Even Parity of Parallel Surfaces at Non-Y Cubes (Figure 10.6b, generalizes Figure 9.9d-g). There should be an even number of correlation surface pieces parallel to the normal

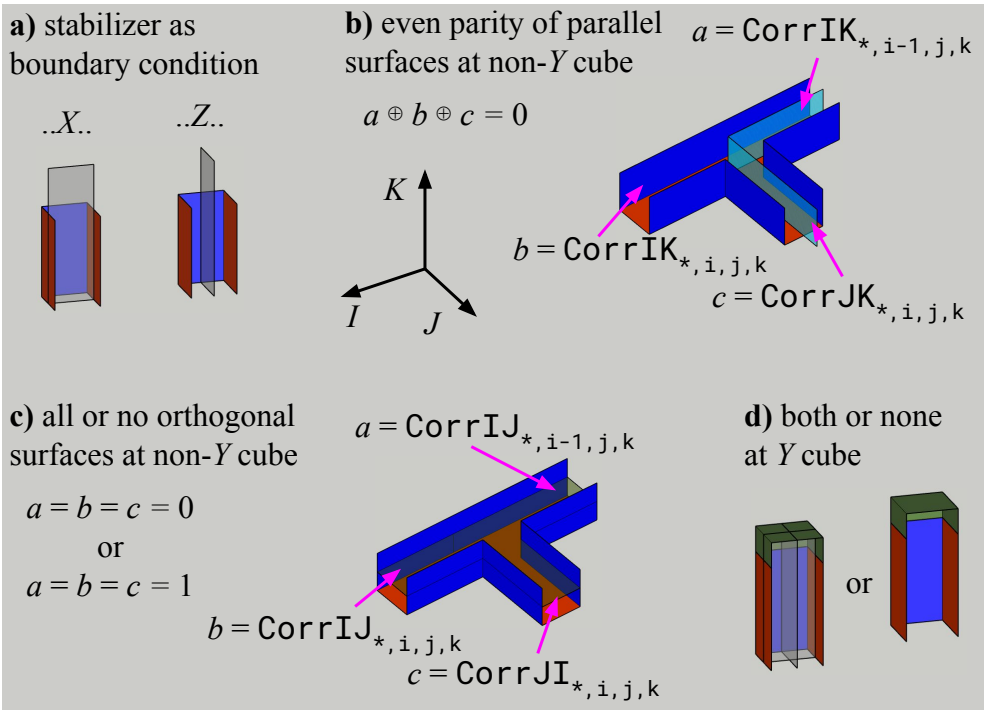


Figure 10.6: Functionality constraints in LaSsynth. **a)** At a port, the correlation surface should be consistent with the stabilizer: only connecting red faces if X; only connecting blue faces if Z; both if Y; neither if I. **b)** An even number of the correlation surfaces parallel to the normal direction should be present. **c)** All or none of the correlation surfaces orthogonal to the normal direction should be present. **d)** Both or no correlation surfaces should be present at a Y cube.

direction of the cube. In Figure 10.5, the normal direction for $(0, 1, 2)$ is J, so

$$\begin{aligned}\Big[\ \overline{\texttt{YCube}_{0,1,2}}\ \wedge \overline{\texttt{ExistJ}_{0,0,2}}\ \wedge \overline{\texttt{ExistJ}_{0,1,2}}\ \Big] \Rightarrow \Big\{\big[\texttt{ExistI}_{0,1,2}\ \wedge \texttt{CorrIJ}_{1,0,1,2}\big] \oplus \\ \big[\texttt{ExistK}_{0,1,1}\ \wedge\ \texttt{CorrKJ}_{1,0,1,1}\big] \oplus \big[\texttt{ExistK}_{0,1,2} \wedge \texttt{CorrKJ}_{1,0,1,2}\big] = 0\Big\},\end{aligned} \tag{10.10}$$

where the first and third term on the right hand side are 1 because the said correlation surface pieces exist, and the second term is 0 because there is no correlation surface in pipe from $(0, 1, 1)$ to $(0, 1, 2)$. Although it is in general costly to express parity operator with first-order logic, we are only dealing with three or four terms, so the translation is simple.

All or No Orthogonal Surfaces at Non-Y Cubes (Figure 10.6c, generalizes Figure 9.9b-c). The correlation surface pieces orthogonal to the normal direction at a cube should all be present or all missing. Let us consider $(1, 0, 1)$ with normal direction I:

$$\begin{aligned}\Big[\ \overline{\texttt{YCube}_{1,0,1}}\ \wedge \overline{\texttt{ExistI}_{0,0,1}}\ \wedge \overline{\texttt{ExistI}_{1,0,1}}\ \Big] \Rightarrow\ \Big\{\Big[\ \overline{\texttt{ExistJ}_{1,0,1}}\ \vee\ \texttt{CorrJK}_{1,1,0,1}\Big] \wedge \\ \Big[\ \overline{\texttt{ExistK}_{1,0,0}} \vee \texttt{CorrKJ}_{1,1,0,0}\Big] \wedge \Big[\ \overline{\texttt{ExistK}_{1,0,1}}\ \vee \texttt{CorrKJ}_{1,1,0,1}\Big]\Big\} \bigvee \Big\{\Big[\ \overline{\texttt{ExistJ}_{1,0,1}} \vee \\ \overline{\texttt{CorrJK}_{1,1,0,1}}\ \Big] \wedge \Big[\ \overline{\texttt{ExistK}_{1,0,0}}\ \vee \overline{\texttt{CorrKJ}_{1,1,0,0}}\ \Big] \wedge \Big[\ \overline{\texttt{ExistK}_{1,0,1}}\ \vee \overline{\texttt{CorrKJ}_{1,1,0,1}}\ \Big]\Big\},\end{aligned} \tag{10.11}$$

where we have two options on the right hand side corresponding to either correlation surface pieces are all present or all missing. In each option, there are three terms corresponding to the three possible pipes connecting $(1, 0, 1)$. In each term, if the `Exist` variable is 0, the term trivially evaluates to 1. This corresponds to the fact that if a pipe does not exist, we do not need to consider correlation surface variables inside it.

10.2 Software Implementation

The structure of our LaS synthesizer, LaSsynth, is exhibited in Figure 10.7a. Given an input file following the specification in Figure 10.1b, we add variables and constraints in the formulation to an SMT (satisfiability modulo theories) model in Z3 SMT solver [dB08]. The model can be solved directly in Z3, but in our experience, the internal solver does not offer the best performance. Thus, while keeping the option of solving the model directly, we

support using Z3 just to simplify the model and transform it to a SAT instance stored in the standard SAT format–DIMACS. Then, we use another SAT solver, Kissat [BF22], to solve it. Since DIMACS is the standard format, it is straightforward to port to any SAT solver on the market with minimal code changes. We chose to still keep Z3 in the loop because some of its simplification 'tactics', e.g., `simplify` and `propagate-values` make a big difference to later SAT solving in our experience. After Kissat solves the SAT instance, we return to Z3, set the variable values according to the SAT result, and let Z3 solve the model again. This second solving is negligible compared to the first one since Z3 is just re-deriving the variables it simplified away previously. In our experiment, it never goes on more than 1 s.

At this point, we have derived the values for all the variables in the formulation. However, given how we have formulated the problem, the SAT solver has no preference for empty space, so the solution found may contain structures that are valid but unnecessary. As post-processing, if a cube is not connected to any ports, it can be pruned away because it has no effect on the functionality of the LaS. Usually, what the pruning removes are pipe "donuts" isolated from all other pipes. Another post-processing step is coloring the K-pipes, i.e., deriving two arrays of additional values, `ColorKP` and `ColorKM`, for the color of K-pipe at the upper and lower ends. If a K-pipe end is "dangling", it must be a port, then its color is given in the input. If it touches any I- or J-pipes, its color can be inferred following the color matching constraints. Finally, if it is in a K passthrough, we can always add domain walls to legalize any K-pipe with different colors at two ends.

LaSsynth has three kinds of output. All the variable assignments constitute our textual LaS representation, LaSre. The second output are 3D modeling files in `gltf` format. In fact, many of the pipe diagrams in the previous and current chapters are rendered using `gltf` files generated by LaSsynth. We can also generate the 3D model with a specific correlation surface like Figure 10.5. Finally, LaSsynth can induce a ZX diagram of the LaS. It uses Stim ZX [Gid21] to derive the stabilizers of this ZX diagram, and compare these stabilizers with the ones given in the input for verification.

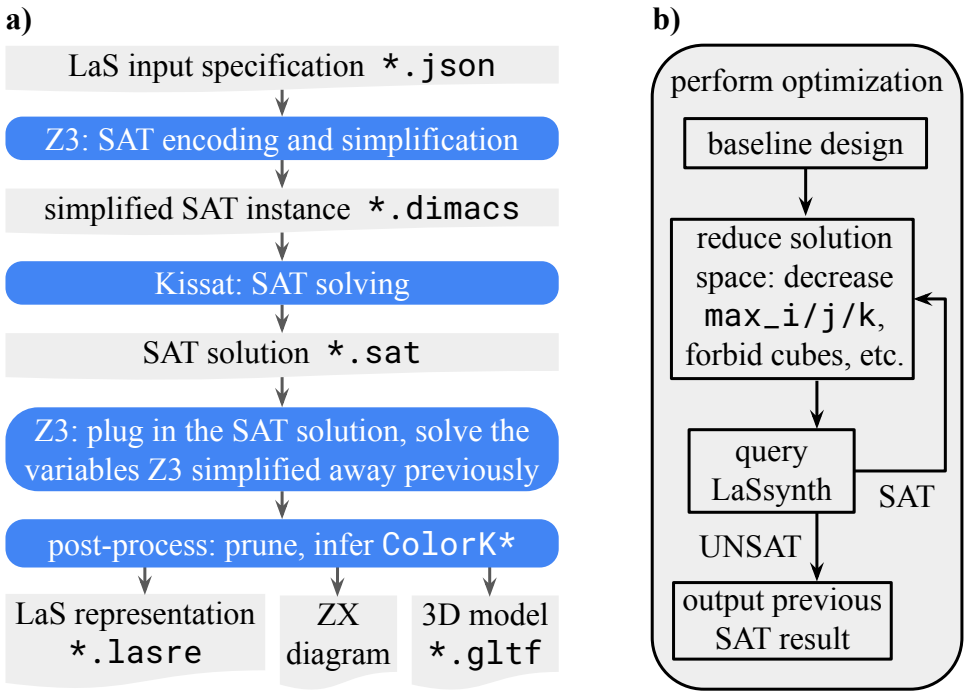


Figure 10.7: Software implementation of LaSsynth. **a)** Workflow of LaSsynth. **b)** Performing LaS optimizations with LaSsynth.

In summary, LaSsynth consumes a specification file and either asserts it is unsatisfiable or produces a LaS satisfying the input. To perform *optimization*, we need to query LaSsynth multiple times as illustrated by Figure 10.7b. If there is a known LaS design, we can treat it as the baseline and revise the specification to reduce the solution space in search for better designs. The simplest example is shrinking the allowed volume by decreasing `max_i/j/k`. We also provide the interface to forbid certain cubes by setting all the `Exist` variables of its pipes to 0, which can be useful if the user is enforcing more complex shapes. The user can also define more general techniques because we provide the interface to set the value of an arbitrary variable in the SMT model to a user-provided value. After the solution space reduces, we query LaSsynth again, until there is no solution, by which we know that the optimal solution is the last satisfying solution. This approach is descending in the sense that we start from a higher-volume solution and gradually find lower-volume solutions. A

drawback of this approach is that sometimes the baseline design is too bad, and the SAT solving takes a very long time given the unnecessarily large dimensions. Thus, sometimes it also helps to take an ascending approach where we start at unsatisfiable settings, e.g., a very small `max_i/j/k`, and grow the solution space until LaSsynth returns a solution.

To exploit certain flexibility in the problem, we may also query LaSsynth many times to search for designs, but not necessarily changing the size of the solution space. For example, in the T-factory later discussed, many ports are functionally symmetric, but when we lay them out in the 3D space, some symmetries are broken. This means, even with the same allowed volume, some permutations of the ports may be satisfiable while others are not. In this case, we can generate a specification file for each promising permutation and run many LaSsynth jobs in parallel. Other than the order of the ports, the location of the ports can also be flexible sometimes. Again, we can run many LaSsynth jobs in parallel, one for each possibility. We have not implemented any general interface to perform explorations under these flexibilities because it greatly depends on the properties of the problem the user has.

10.3 Evaluation

LaSsynth is open source as an optional component in Stim.[1] It is implemented in Python3 with a dependency on Z3 4.12.1.0 [dB08]. Installing Kissat 3.1.0 [BF22] is required for the recommended SAT solving path. Installing Stim ZX [Gid21] is optional for verifying the stabilizers of LaS. The following evaluations are done on a Linux server with two AMD EPYC 7V13 Processor and 512GB DRAM. We provide a summary of results before diving into experimental details.

[1] https://github.com/quantumlib/Stim/tree/main/glue/lattice_surgery

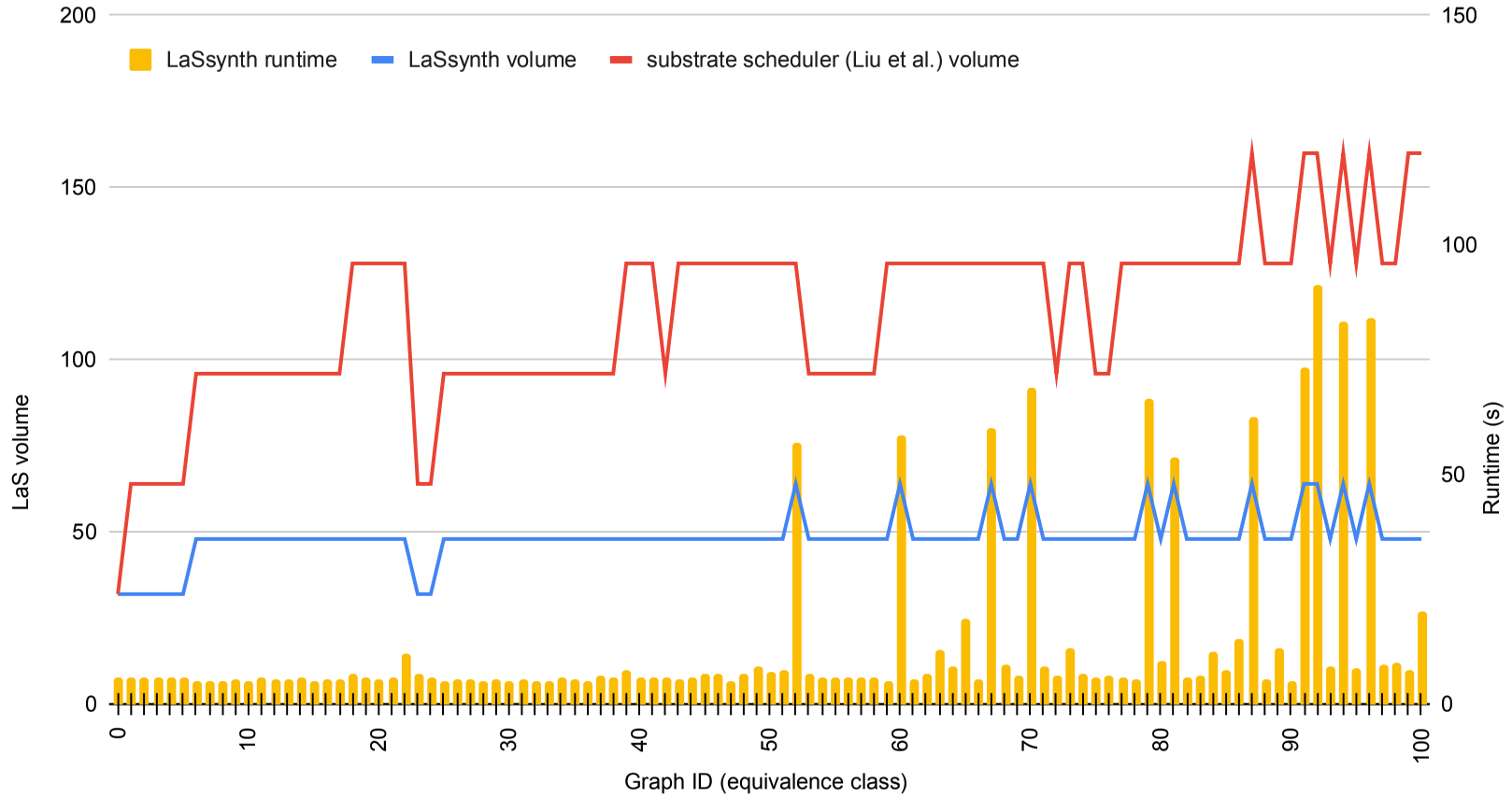


Figure 10.8: LaS volume and LaSsynth runtime for 8-qubit graph state generation. Each graph represents a local-unitary equivalence class of graph states.

Generic graph state generation provides a representative scenario for the intended use of LaSsynth. A compiler in [LBM23] is tailored for a (logical) 2-lane architecture with careful qubit initialization and optimal gate scheduling. LaSsynth outperforms this baseline by 56% on average across a comprehensive 8-qubit graph state benchmark set, as shown in Figure 10.8. Our advantage lies in the smaller footprint per logical qubit, coupled with the ability to establish more intricate connectivity for accessing them.

We can utilize LaSsynth to construct non-Clifford LaS by inputting non-Clifford resources from ports. For instance, in the majority gate, an important LaS in Shor's algorithm, three ports consume a $|\mathrm{CCZ}\rangle$. LaSsynth reduces volume by 40% compared to the design in [GF19b]. The corresponding ZX diagram is challenging for human understanding because of the creative use of generalized Hopf rule (see Figure 9.11c). Our ZX calculus verification reaffirms the correctness of our result and actually reveals the error of the design in [GF19b].

We leverage LaSsynth to optimize T-factories, the dominating cost in FTQC. There are some nuanced considerations at the non-Clifford input ports because of nondeterministic state injections. Although we choose to get around these intricacies with very basic techniques, LaSsynth still discovers a 15-to-1 T-factory, showcased in Figure 10.9, 8% smaller than state-of-the-art design [FG19, GF19a]. If neglecting state injection delays, LaSsynth discovers a design, as depicted in Figure 10.10b, 18% smaller than the state-of-the-art in this setting [Lit19a], using a smaller footprint while maintaining the same depth.

The primary use case for LaSsynth is optimizing critical subroutines. Despite potential exponential scaling of the internal SAT solving, it consistently outperforms human expert designs at the scale of realistically significant subroutines, as evidenced by the presented results. In essence, our advantage lies in more flexibility of allocating, moving, and recycling code patches. As pipe diagrams, the human designs usually let the qubits stay put, i.e., forming "pillars" that vertically goes through the LaS, and lattice surgery is performed by unit-time horizontal crossbars connecting to the pillars. In contrast, the vertical pipes in our generated LaS can terminate and begin at will, so the solver explores a much larger design

space, e.g., ancillas may not be immediately recycled, but squeezed and moved around to interact with other qubits.

10.3.1 Methodology of Graph State Generation Evaluation

Hardware assumption. Aligning with the baseline [LBM23], there are 2 lanes of surface code tiles available as workspace, each tile is $d \times d$ physical qubits. In our specification, this means limiting `max_j` $= 2$. Figure 9.10b provides an illustration, where the back lane is for (logical) qubit output, and the front lane is ancillary.

Baseline approach. Liu et al. developed a compiler [LBM23] based on [Lit19a]: initializing logical qubits in selective basis and then performing multi-qubit parity measurements using lattice surgery. They observed that selecting the initialization basis is a Maximum Independent Set problem (MIS). Given the initialization in Figure 9.10c, only the last two stabilizers in Figure 9.10a require measurement. MIS is NP-hard, so, to be fair, *we gave this compiler the same amount of time as LaSsynth spends.* In their setting, to enable measurements in both the X and Z bases, the qubits must expose both types of boundaries to the ancilla lane, necessitating 2-tile patches (on the right side of Figure 9.10c), pushing footprint of their subroutine to 16×2=32.

Comprehensive benchmark set. There are $2^{n(n-1)/2}$ graphs for n nodes, but many of them are equivalent up to single-qubit Cliffords. Our benchmarks are 101 graphs from a database [CDL11] representing *all* the equivalence classes of 8-qubit graphs.

Our approach. Because of the 2-lane assumption, we specify 8×2 footprint along with the graph state stabilizers to LaSsynth. We initiate the search at a depth of 3 and iteratively adjust it based on the response—increasing depth if UNSAT or decreasing it if SAT—to determine the optimal depth. The resulting LaS volume is then 8×2× the optimal depth.

Source of advantage. The baseline approach relies on 2-tile patches, doubling the required footprint and placing it at a disadvantage. This necessity stems from the limitation of 1-tile

patches, where only one basis is exposed to the ancilla at a time, posing a challenge for human intuition to access all required bases effectively. LaSsynth overcomes this limitation by employing domain walls (depicted as yellow rings) and intricate connectivity, as seen in Figure 9.10b. In contrast, the baseline solutions would consist only of horizontal bars, i.e., parity measurements, connecting to some of the 8 qubits.

10.3.2 Methodology of Majority Gate Evaluation

Port requirements of the majority gate. To align with the use case in [GF19b] (see Figure 9.11a), some port location requirements must be met. Specifically, t and t' must be at the same height, so do a and a', and c_{in} and c_{out}. The routed $|\text{CCZ}\rangle$ necessitates three additional vertically aligned ports above them. These constraints imply `max_j` ≥ 3 and `max_k` ≥ 5, leaving the only dimension that can shrink as I.

Importance of verification. To verify a design, LaSsynth extracts a ZX diagram, and leverages Stim ZX to derive its stabilizers. When we read off the 5×3×5 design in [GF19b] to a LaSre and give it to LaSsynth, verification fails, underscoring the susceptibility of human designed LaS to errors and the practical challenges of manual verification, even for experts.

10.3.3 Methodology of T-Factory Evaluation

Fixups. When injecting $|T\rangle$ in the surface code, half the time yields $|T\rangle$, and the other half yields $|T^\dagger\rangle$ [FG19]. Consequently, we may need to apply an S gate via Y cubes based on the injection result. These dynamic cubes are not included in the formulation since they depend on *runtime* information. We need to reserve space in the LaS for these cubes. How much space and where is necessary to accommodate any of the 2^{15} injection outcomes is an involved topic. For simplicity, we adopt a straightforward technique illustrated in Figure 10.9: each injection connects to a K-pipe and then bends inward to the rest of the pipe diagram. Fixups for each injection can be attached where the pipe bends. We do not include the fixups to

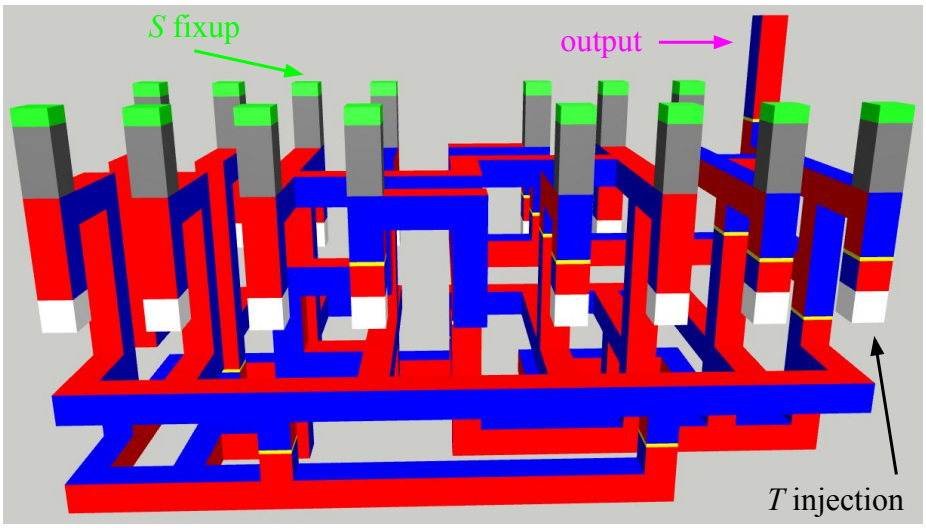


Figure 10.9: Pipe diagram of a 15-to-1 T-factory with $9 \times 4 \times 4.5$ spacetime volume generated with LaSsynth. White boxes are magic state injections. An S fixup consists of conditional K-pipes (gray) and Y cubes (green).

the specification and append the fixup layer after synthesis.

T-factory optimization results. The baseline design actually employs *half-distance rotation* of tiles, which is not available in our formulation. Despite this disadvantage, by iteratively calling LaSsynth with shrinking volume, we obtained solutions with volumes of $7\times5\times5=175$ and $6\times6\times4.5=9\times4\times4.5=162$. In Figure 10.9, we showcase one of the best designs, 8% smaller than the baseline. We verified its ZX diagram using Stim ZX. Notably, by avoiding half-distance rotations present in the baseline, this design opens opportunities for using half-distance elsewhere. Thus, it is of interest to quantum error correction experts to explore how to reduce the distance in regions of this design, potentially achieving further improvements.

T-factory assuming no classical delay. Much of the preceding discussion delves into fixup details. Ignoring classical injection delay, [Lit19a] presents a 121-volume factory design utilizing 11 patches (Figure 10.10c top) and a depth of 11, with four injections from the bottom and the remainder from the side (Figure 10.10a). Under the same assumption, LaSsynth

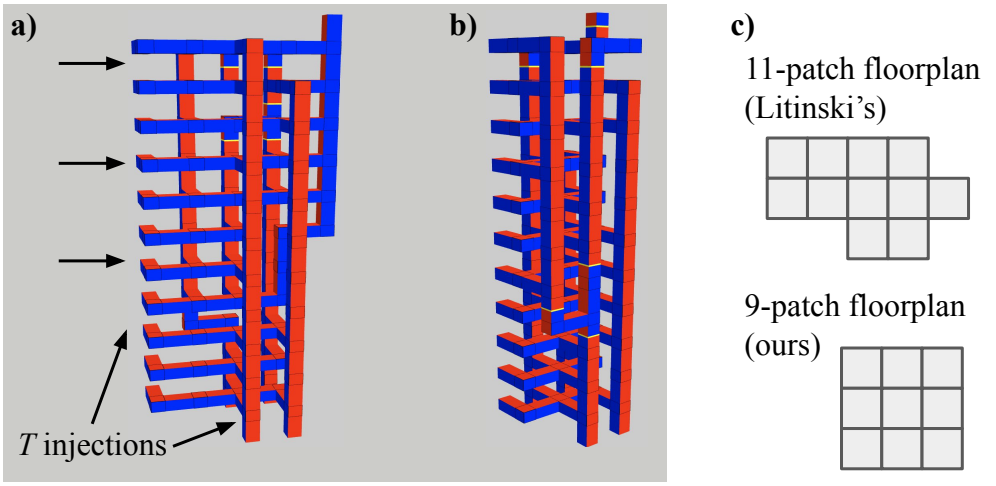


Figure 10.10: T-factory assuming no classical delay for injections. **a)** A generated design with volume 121, same as [Lit19a]. **b)** Optimized design with 3×3×11 volume. **c)** Floorplan comparison of the two designs.

derives a design (Figure 10.10b) with volume 3×3×11=99, achieving an 18% reduction by using a smaller footprint (Figure 10.10c bottom) than the 121-volume design.

10.3.4 Observations on Runtime

Scalability metrics. The runtime of LaSsynth may scale unfavorably due to its dependence on SAT solving. However, its target use case is frequently used subroutines rather than entire algorithms, and the presented results demonstrate its effectiveness in solving significant and realistic subroutines. Some runtimes are recorded in Table 10.1, where the formulation yields a scaling factor, Vn_{stab}, i.e., the volume times the number of stabilizers. It is worth noting that this column may appear slightly larger than expected because of padding layers on boundaries to support ports or non-rectangular floorplans. While the 121-factory has a larger Vn_{stab} than the 162-factory, it is a simpler problem to solve. A better indicator is the number of variables and clauses of the generated SAT CNF.

Random seed: diversity matters. We employed 10 random seeds on the same CNF, and

Table 10.1: Size and runtime of LaSsynth for presented non-Clifford designs

Problem	Vn_{stab}	**Variables**	**Clauses**	**Min Time (s)**	**Standard Deviation**
Majority	720	8173	56851	9.02	0.33
99-factory	1728	33650	248974	20.6	0.61
121-factory	2880	35657	267544	40.9	8.3
162-factory	2304	43070	326305	469	4E3

the standard deviations of the runtimes are presented in the last column of Table 10.1. Notably, for the 162-factory, the runtime difference can be as much as 26 times, indicating that multiple SAT solvers with different seeds may significantly expedite finding a solution, justifying the use of portfolio-based SAT solving, e.g., [SS21].

To UNSAT or not, it is a question. Unsatisfiable specifications took longer than satisfiable ones. In Figure 10.8, it is noticeable that long runtimes consistently accompany a 'spike' in volume. These instances represent cases where LaSsynth initiates with a depth of 3, requires a relatively lengthy period to determine unsatisfiability, and subsequently discovers a solution with depth 4. Generally, this is the price to pay for the optimality guarantee. For scenarios where optimal solutions are not strictly necessary, prioritizing satisfiability initially—such as with incomplete approaches like MaxSAT, or iterative shrinking a design while retaining learned information—can be more beneficial in obtaining good designs efficiently.

10.4 Related Works

[HND17] first provided a compiler that takes in ICM (initialization, CNOT, and measurement) representation and translates the gates to lattice surgery operations. Later on, researchers opt for a more efficient operation with lattice surgery, multi-qubit Pauli measure-

ments. Thus, a few works focused on implementing FTQC on a 2D grid of qubits with the multi-qubit-Pauli-based gate set [BMT22, CC22, FG19, Lit19a]. [LN22] still used this gate set but discussed the advantage of having non-local connectivity. In terms of software, [PF20] provided an instruction set for this gate set and [WNS23] provided a compiler. However, as we demonstrated above, it is beneficial to consider optimizations beyond this gate set.

Some previous works focused on improving specific components, not generic quantum circuits. Since the magic state factories take up a lot of volume, they have become a natural target for such optimizations. [Lit19b] further developed the aforementioned technique of selectively reducing code distances, which can be applied in combination with the optimizations we present in this chapter. [GF19a] considered the interplay of T-factories with $|\mathrm{CCZ}\rangle$-factories and presented improved factory designs. [GF19b] provided further improvements on $|\mathrm{CCZ}\rangle$-factories. However, all these works are manual efforts instead of an automated synthesizer.

There is a similar line of works for defect based FTQC on surface codes [FMM12]. The compilation problem is formulated as routing FTQC components [PDF16]. After a manual approach of bridge compression was proposed [FD13], researchers encoded the problem to integer linear programming [HLT21, LYL18], which is another kind of mathematical programming than SAT. Heuristic compilation approaches have been presented for optimizing communication [HCJ21, JGH17, TMN17], or for T-factory [DHJ18]. However, defect-based computation is phasing out because of higher overheads than lattice surgery [Lit19a, FG19].

[SC22] utilizes an SMT solver to synthesize fault-tolerant Clifford circuit in a bottom-up fashion like this chapter. However, the gate set consists of CNOT, X, Y, Z, S, and H, quite different from generic lattice surgery operations in this chapter.

CHAPTER 11

Conclusion and Outlook

This dissertation focuses on layout synthesis for quantum computing, a critical compilation task that transforms quantum programs to adhere to the connectivity constraints of various quantum computing architectures. It covers layout synthesis for three types of architectures: those with static coupling graphs, dynamically field-programmable qubit arrays based on neutral atoms, and a fault-tolerant architecture using 2D surface codes. Each of these formulations has spurred distinct lines of research and development.

Development of SMT-based Tools. Converting layout synthesis problems into satisfiability modulo theories models has led to the creation of specialized tools such as OLSQ, OLSQ-DPQA, and LaSsynth. These tools are designed to find optimal solutions on limited scale layout synthesis problems. They prove particularly useful in cases where optimality is critical, albeit within the constraints of smaller, more manageable problem sizes.

Insights Leading to Scalable Compilers. The formulations we invent provide valuable insights into the architectural constraints, facilitating the development of efficient and high-quality compilers like Enola. These compilers can handle large-scale layout synthesis problems, addressing the need for broader application scopes.

Benchmark Construction for Tool Evaluation. The formulation process has also enabled the construction of benchmarks with specific properties, such as QUEKO benchmarks, which possess known optimal solutions. These benchmarks are instrumental in measuring and enhancing the effectiveness and optimality of various layout synthesis tools, providing a standard against which tool performance can be rigorously evaluated.

Future directions point to the necessity of further development in layout synthesis, especially for architectures based on neutral atoms which are characterized by their highly dynamic nature. As more levels of abstraction are integrated into fault-tolerant quantum computing, the complexity and scope of layout synthesis are also expected to expand.

[AAA21] MD SAJID ANIS, Héctor Abraham, AduOffei, Rochisha Agarwal, Gabriele Agliardi, Merav Aharoni, Ismail Yunus Akhalwaya, Gadi Aleksandrowicz, Thomas Alexander, Matthew Amy, Sashwat Anagolum, Eli Arbel, Abraham Asfaw, Anish Athalye, Artur Avkhadiev, Carlos Azaustre, PRATHAMESH BHOLE, Abhik Banerjee, Santanu Banerjee, Will Bang, Aman Bansal, Panagiotis Barkoutsos, Ashish Barnawal, George Barron, George S. Barron, Luciano Bello, Yael Ben-Haim, Daniel Bevenius, Dhruv Bhatnagar, Arjun Bhobe, Paolo Bianchini, Lev S. Bishop, Carsten Blank, Sorin Bolos, Soham Bopardikar, Samuel Bosch, Sebastian Brandhofer, Brandon, Sergey Bravyi, Nick Bronn, Bryce-Fuller, David Bucher, Artemiy Burov, Fran Cabrera, Padraic Calpin, Lauren Capelluto, Jorge Carballo, Ginés Carrascal, Adam Carriker, Ivan Carvalho, Adrian Chen, Chun-Fu Chen, Edward Chen, Jielun (Chris) Chen, Richard Chen, Franck Chevallier, Kartik Chinda, Rathish Cholarajan, Jerry M. Chow, Spencer Churchill, Christian Claus, Christian Clauss, Caleb Clothier, Romilly Cocking, Ryan Cocuzzo, Jordan Connor, Filipe Correa, Abigail J. Cross, Andrew W. Cross, Simon Cross, Juan Cruz-Benito, Chris Culver, Antonio D. Córcoles-Gonzales, Navaneeth D, Sean Dague, Tareq El Dandachi, Animesh N Dangwal, Jonathan Daniel, Marcus Daniels, Matthieu Dartiailh, Abdón Rodríguez Davila, Faisal Debouni, Anton Dekusar, Amol Deshmukh, Mohit Deshpande, Delton Ding, Jun Doi, Eli M. Dow, Eric Drechsler, Eugene Dumitrescu, Karel Dumon, Ivan Duran, Kareem EL-Safty, Eric Eastman, Grant Eberle, Amir Ebrahimi, Pieter Eendebak, Daniel Egger, ElePT, Emilio, Alberto Espiricueta, Mark Everitt, Davide Facoetti, Farida, Paco Martín Fernández, Samuele Ferracin, Davide Ferrari, Axel Hernández Ferrera, Romain Fouilland, Albert Frisch, Andreas Fuhrer, Bryce Fuller, MELVIN GEORGE, Julien Gacon, Borja Godoy Gago, Claudio Gambella, Jay M. Gambetta, Adhisha Gammanpila, Luis Garcia, Tanya Garg, Shelly Garion, Tim Gates, Leron Gil, Austin Gilliam, Aditya Giridharan, Juan Gomez-Mosquera, Gonzalo, Salvador de la Puente González, Jesse Gorzinski, Ian Gould, Donny Greenberg, Dmitry Grinko, Wen Guan, John A. Gunnels, Harshit Gupta, Naman Gupta, Jakob M. Günther, Mikael Haglund, Isabel Haide, Ikko Hamamura, Omar Costa Hamido, Frank Harkins, Areeq Hasan, Vojtech Havlicek, Joe Hellmers, Łukasz Herok, Stefan Hillmich, Hiroshi Horii, Connor Howington, Shaohan Hu, Wei Hu, Junye Huang, Rolf Huisman, Haruki Imai, Takashi Imamichi, Kazuaki Ishizaki, Ishwor, Raban Iten, Toshinari Itoko, Alexander Ivrii, Ali Javadi, Ali Javadi-Abhari, Wahaj Javed, Qian Jianhua, Madhav Jivrajani, Kiran Johns, Scott Johnstun, Jonathan-Shoemaker, JosDenmark, JoshDumo, John Judge, Tal Kachmann, Akshay Kale, Naoki Kanazawa, Jessica Kane, Kang-Bae, Annanay Kapila, Anton Karazeev, Paul Kassebaum, Josh Kelso, Scott Kelso, Vismai Khanderao,

Spencer King, Yuri Kobayashi, Kovi11Day, Arseny Kovyrshin, Rajiv Krishnakumar, Vivek Krishnan, Kevin Krsulich, Prasad Kumkar, Gawel Kus, Ryan LaRose, Enrique Lacal, Raphaël Lambert, Haggai Landa, John Lapeyre, Joe Latone, Scott Lawrence, Christina Lee, Gushu Li, Jake Lishman, Dennis Liu, Peng Liu, Yunho Maeng, Saurav Maheshkar, Kahan Majmudar, Aleksei Malyshev, Mohamed El Mandouh, Joshua Manela, Manjula, Jakub Marecek, Manoel Marques, Kunal Marwaha, Dmitri Maslov, Paweł Maszota, Dolph Mathews, Atsushi Matsuo, Farai Mazhandu, Doug McClure, Maureen McElaney, Cameron McGarry, David McKay, Dan McPherson, Srujan Meesala, Dekel Meirom, Corey Mendell, Thomas Metcalfe, Martin Mevissen, Andrew Meyer, Antonio Mezzacapo, Rohit Midha, Daniel Miller, Zlatko Minev, Abby Mitchell, Nikolaj Moll, Alejandro Montanez, Gabriel Monteiro, Michael Duane Mooring, Renier Morales, Niall Moran, David Morcuende, Seif Mostafa, Mario Motta, Romain Moyard, Prakash Murali, Jan Müggenburg, David Nadlinger, Ken Nakanishi, Giacomo Nannicini, Paul Nation, Edwin Navarro, Yehuda Naveh, Scott Wyman Neagle, Patrick Neuweiler, Aziz Ngoueya, Johan Nicander, Nick-Singstock, Pradeep Niroula, Hassi Norlen, NuoWenLei, Lee James O'Riordan, Oluwatobi Ogunbayo, Pauline Ollitrault, Tamiya Onodera, Raul Otaolea, Steven Oud, Dan Padilha, Hanhee Paik, Soham Pal, Yuchen Pang, Ashish Panigrahi, Vincent R. Pascuzzi, Simone Perriello, Eric Peterson, Anna Phan, Francesco Piro, Marco Pistoia, Christophe Piveteau, Julia Plewa, Pierre Pocreau, Alejandro Pozas-Kerstjens, Rafał Pracht, Milos Prokop, Viktor Prutyanov, Sumit Puri, Daniel Puzzuoli, Jesús Pérez, Quant02, Quintiii, Isha R, Rafey Iqbal Rahman, Arun Raja, Roshan Rajeev, Nipun Ramagiri, Anirudh Rao, Rudy Raymond, Oliver Reardon-Smith, Rafael Martín-Cuevas Redondo, Max Reuter, Julia Rice, Matt Riedemann, Rietesh, Drew Risinger, Marcello La Rocca, Diego M. Rodríguez, RohithKarur, Ben Rosand, Max Rossmannek, Mingi Ryu, Tharrmashastha SAPV, Nahum Rosa Cruz Sa, Arijit Saha, Abdullah Ash-Saki, Sankalp Sanand, Martin Sandberg, Hirmay Sandesara, Ritvik Sapra, Hayk Sargsyan, Aniruddha Sarkar, Ninad Sathaye, Bruno Schmitt, Chris Schnabel, Zachary Schoenfeld, Travis L. Scholten, Eddie Schoute, Mark Schulterbrandt, Joachim Schwarm, James Seaward, Sergi, Ismael Faro Sertage, Kanav Setia, Freya Shah, Nathan Shammah, Rohan Sharma, Yunong Shi, Jonathan Shoemaker, Adenilton Silva, Andrea Simonetto, Divyanshu Singh, Parmeet Singh, Phattharaporn Singkanipa, Yukio Siraichi, Siri, Jesús Sistos, Iskandar Sitdikov, Seyon Sivarajah, Magnus Berg Sletfjerding, John A. Smolin, Mathias Soeken, Igor Olegovich Sokolov, Igor Sokolov, Vicente P. Soloviev, SooluThomas, Starfish, Dominik Steenken, Matt Stypulkoski, Adrien Suau, Shaojun Sun, Kevin J. Sung, Makoto Suwama, Oskar Słowik, Hitomi Takahashi, Tanvesh Takawale, Ivano Tavernelli, Charles Taylor, Pete Taylour, Soolu Thomas, Kevin Tian, Mathieu Tillet, Maddy Tod, Miroslav Tomasik, Caroline Tornow, Enrique de la Torre, Juan Luis Sánchez Toural, Kenso Trabing, Matthew Treinish, Dimitar Trenev, TrishaPe, Felix Truger, Georgios

Tsilimigkounakis, Davindra Tulsi, Wes Turner, Yotam Vaknin, Carmen Recio Valcarce, Francois Varchon, Adish Vartak, Almudena Carrera Vazquez, Prajjwal Vijaywargiya, Victor Villar, Bhargav Vishnu, Desiree Vogt-Lee, Christophe Vuillot, James Weaver, Johannes Weidenfeller, Rafal Wieczorek, Jonathan A. Wildstrom, Jessica Wilson, Erick Winston, WinterSoldier, Jack J. Woehr, Stefan Woerner, Ryan Woo, Christopher J. Wood, Ryan Wood, Steve Wood, James Wootton, Matt Wright, Lucy Xing, Bo Yang, Daniyar Yeralin, Ryota Yonekura, David Yonge-Mallo, Ryuhei Yoshida, Richard Young, Jessie Yu, Lebin Yu, Christopher Zachow, Laura Zdanski, Helena Zhang, Christa Zoufal, aeddins ibm, alexzhang13, b63, bartek bartlomiej, bcamorrison, brandhsn, charmerDark, deeplokhande, dekel.meirom, dime10, dlasecki, ehchen, fanizzamarco, fs1132429, gadial, galeinston, georgezhou20, georgios ts, gruu, hhorii, hykavitha, itoko, jessica angel7, jliu45, jscott2, klinvill, krutik2966, ma5x, michelle4654, msuwama, ntgiwsvp, ordmoj, sagar pahwa, pritamsinha2304, ryancocuzzo, saswati qiskit, septembrr, sethmerkel, shaashwat, sternparky, strickroman, tigerjack, tsura crisaldo, vadebayo49, welien, willhbang, wmurphy collabstar, yang.luh, and Mantas Čepulkovskis. "Qiskit: An Open-source Framework for Quantum Computing." Zenodo:2573505, 2021, DOI:10.5281/zenodo.2573505.

[AAB19] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando G. S. L. Brandao, David A. Buell, Brian Burkett, Yu Chen, Zijun Chen, Ben Chiaro, Roberto Collins, William Courtney, Andrew Dunsworth, Edward Farhi, Brooks Foxen, Austin Fowler, Craig Gidney, Marissa Giustina, Rob Graff, Keith Guerin, Steve Habegger, Matthew P. Harrigan, Michael J. Hartmann, Alan Ho, Markus Hoffmann, Trent Huang, Travis S. Humble, Sergei V. Isakov, Evan Jeffrey, Zhang Jiang, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Paul V. Klimov, Sergey Knysh, Alexander Korotkov, Fedor Kostritsa, David Landhuis, Mike Lindmark, Erik Lucero, Dmitry Lyakh, Salvatore Mandrà, Jarrod R. McClean, Matthew McEwen, Anthony Megrant, Xiao Mi, Kristel Michielsen, Masoud Mohseni, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Murphy Yuezhen Niu, Eric Ostby, Andre Petukhov, John C. Platt, Chris Quintana, Eleanor G. Rieffel, Pedram Roushan, Nicholas C. Rubin, Daniel Sank, Kevin J. Satzinger, Vadim Smelyanskiy, Kevin J. Sung, Matthew D. Trevithick, Amit Vainsencher, Benjamin Villalonga, Theodore White, Z. Jamie Yao, Ping Yeh, Adam Zalcman, Hartmut Neven, and John M. Martinis. "Quantum supremacy using a programmable superconducting processor." *Nature*, **574**(7779):505–510, 2019, DOI:10.1038/s41586-019-1666-5.

[AAG19] Abdullah Ash-Saki, Mahabubul Alam, and Swaroop Ghosh. "QURE: Qubit Reallocation in Noisy Intermediate-Scale Quantum Computers." In *Proceedings of the 56th Annual Design Automation Conference*, DAC '19, Las Vegas, NV, USA, 2019. ACM Press, DOI:10.1145/3316781.3317888.

[AB08] Dorit Aharonov and Michael Ben-Or. "Fault-Tolerant Quantum Computation with Constant Error Rate." *SIAM Journal on Computing*, **38**(4):1207–1282, 2008, DOI:10.1137/S0097539799359385.

[AG04] Scott Aaronson and Daniel Gottesman. "Improved simulation of stabilizer circuits." *Physical Review A*, **70**:052328, 2004, DOI:10.1103/PhysRevA.70.052328.

[ALF17] C. G. Almudever, L. Lao, X. Fu, N. Khammassi, I. Ashraf, D. Iorga, S. Varsamopoulos, C. Eichler, A. Wallraff, L. Geck, A. Kruth, J. Knoch, H. Bluhm, and K Bertels. "The Engineering Challenges in Quantum Computing." In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pp. 836–845, Lausanne, Switzerland, 2017. IEEE, DOI:10.23919/DATE.2017.7927104.

[AMM13] M. Amy, D. Maslov, M. Mosca, and M. Roetteler. "A Meet-in-the-Middle Algorithm for Fast Synthesis of Depth-Optimal Quantum Circuits." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **32**(6):818–830, 2013, DOI:10.1109/TCAD.2013.2244643.

[AVJ98] J Amilhastre, M.C Vilarem, and P Janssen. "Complexity of minimum biclique cover and minimum biclique decomposition for bipartite domino-free graphs." *Discrete Applied Mathematics*, **86**(2-3):125–144, 1998, DOI:10.1016/S0166-218X(98)00039-0.

[BBC95] Adriano Barenco, Charles H. Bennett, Richard Cleve, David P. DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A. Smolin, and Harald Weinfurter. "Elementary gates for quantum computation." *Physical Review A*, **52**(5):3457–3467, 1995, DOI:10.1103/PhysRevA.52.3457. Number: 5.

[BBP21] Sebastian Brandhofer, Hans Peter Büchler, and Ilia Polian. "Optimal Mapping for Near-Term Quantum Architectures based on Rydberg Atoms." In *Proceedings of the 40th IEEE/ACM International Conference on Computer-Aided Design*, Munich, Germany, 2021. DOI:10.1109/ICCAD51958.2021.9643490.

[BDB18] Kyle EC Booth, Minh Do, J Christopher Beck, Eleanor Rieffel, Davide Venturelli, and Jeremy Frank. "Comparing and integrating constraint programming and temporal planning for quantum circuit compilation." In *Twenty-Eighth International Conference on Automated Planning and Scheduling*, 2018, DOI:10.48550/arXiv.1803.06775.

[BDM23] Héctor Bombín, Chris Dawson, Ryan V. Mishmash, Naomi Nickerson, Fernando Pastawski, and Sam Roberts. "Logical Blocks for Fault-Tolerant Topological Quantum Computation." *PRX Quantum*, **4**:020303, 2023, DOI:10.1103/PRXQuantum.4.020303.

[BF22] Armin Biere and Mathias Fleury. "Gimsatul, IsaSAT and Kissat entering the SAT Competition 2022." In Tomas Balyo, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2022 – Solver and Benchmark Descriptions*, volume B-2022-1 of *Department of Computer Science Series of Publications B*, pp. 10–11. University of Helsinki, 2022.

[BH20] Niel de Beaudrap and Dominic Horsman. "The ZX calculus is a language for surface code lattice surgery." *Quantum*, **4**:218, 2020, DOI:10.22331/q-2020-01-09-218.

[BKM18] Adi Botea, Akihiro Kishimoto, and Radu Marinescu. "On the Complexity of Quantum Circuit Compilation." In *Proceedings of the 11th Annual Symposium on Combinatorial Search*, Stockholm, Sweden, 2018. AAAI Press, DOI:10.1609/socs.v9i1.18463.

[BLD21] Jonathan M. Baker, Andrew Litteken, Casey Duckering, et al. "Exploiting Long-Distance Interactions and Tolerating Atom Loss in Neutral Atom Quantum Architectures." In *Proceedings of the 48th Annual International Symposium on Computer Architecture*, ISCA '21, pp. 818–831. IEEE Press, 2021, DOI:10.1109/ISCA52012.2021.00069.

[BLN24] Hector Bombin, Daniel Litinski, Naomi Nickerson, Fernando Pastawski, and Sam Roberts. "Unifying flavors of fault tolerance with the ZX calculus." *Quantum*, **8**:1379, 2024, DOI:10.22331/q-2024-06-18-1379.

[BLP23] Guido Burkard, Thaddeus D. Ladd, Andrew Pan, John M. Nichol, and Jason R. Petta. "Semiconductor spin qubits." *Reviews of Modern Physics*, **95**:025003, 2023, DOI:10.1103/RevModPhys.95.025003.

[BLS22] Dolev Bluvstein, Harry Levine, Giulia Semeghini, Tout T. Wang, Sepehr Ebadi, Marcin Kalinowski, Alexander Keesling, Nishad Maskara, Hannes Pichler, Markus Greiner, Vladan Vuletić, and Mikhail D. Lukin. "A quantum processor based on coherent transport of entangled atom arrays." *Nature*, **604**(7906):451–456, 2022, DOI:10.1038/s41586-022-04592-6.

[BLS24] Dolev Bluvstein, Harry Levine, Giulia Semeghini, Tout T. Wang, Sepehr Ebadi, Marcin Kalinowski, Alexander Keesling, Nishad Maskara, Hannes Pichler, Markus Greiner, Vladan Vuletić, and Mikhail D. Lukin. "Logical quantum processor based on reconfigurable atom arrays." *Nature*, **626**(7997):58–65, 2024, DOI:10.1038/s41586-023-06927-3.

[BMB08] D. Bein, L. Morales, W. Bein, Jr C. O. Shields, Z. Meng, and I. H. Sudborough. "Clustering and the Biclique Partition Problem." In *Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, 2008, DOI:10.1109/HICSS.2008.504.

[BMT22] Michael E. Beverland, Prakash Murali, Matthias Troyer, Krysta M. Svore, Torsten Hoefler, Vadym Kliuchnikov, Guang Hao Low, Mathias Soeken, Aarthi Sundaram, and Alexander Vaschillo. "Assessing requirements to scale to practical quantum advantage." arXiv:2211.07629, 2022, DOI:10.48550/arXiv.2211.07629.

[BPW17] Miriam Backens, Simon Perdrix, and Quanlong Wang. "A Simplified Stabilizer ZX-calculus." *Electronic Proceedings in Theoretical Computer Science*, **236**:1–20, 2017, DOI:10.4204/eptcs.236.1.

[BSA19] Debjyoti Bhattacharjee, Abdullah Ash Saki, Mahabubul Alam, Anupam Chattopadhyay, and Swaroop Ghosh. "MUQUT: Multi-Constraint Quantum Circuit Mapping on NISQ Computers: Invited Paper." In *Proceedings of the 38th IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '19, Westminster, CO, USA, 2019. IEEE, DOI:10.1109/ICCAD45719.2019.8942132.

[BTM07] Jérôme Beugnon, Charles Tuchendler, Harold Marion, Alpha Gaëtan, Yevhen Miroshnychenko, Yvan R. P. Sortais, Andrew M. Lance, Matthew P. A. Jones, Gaétan Messin, Antoine Browaeys, and Philippe Grangier. "Two-dimensional transport and transfer of a single atomic qubit in optical tweezers." *Nature Physics*, **3**(10):696–699, 2007, DOI:10.1038/nphys698.

[CBG21] Arjan Cornelissen, Johannes Bausch, and András Gilyén. "Scalable Benchmarks for Gate-Based Quantum Computers." arXiv:2104.10698, 2021, DOI:10.48550/arXiv.2104.10698.

[CC06] Tung-Chieh Chen and Yao-Wen Chang. "Modern floorplanning based on B*-tree and fast simulated annealing." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **25**(4):637–650, 2006, DOI:10.1109/TCAD.2006.870076.

[CC22] Christopher Chamberland and Earl T. Campbell. "Universal Quantum Computing with Twist-Free and Temporally Encoded Lattice Surgery." *PRX Quantum*, **3**:010331, 2022, DOI:10.1103/PRXQuantum.3.010331.

[CCL19] Iris Cong, Soonwon Choi, and Mikhail D. Lukin. "Quantum Convolutional Neural Networks." *Nature Physics*, **15**(12):1273–1278, 2019, DOI:10.1038/s41567-019-0648-8.

[CCR04] C.-C. Chang, J. Cong, M. Romesis, and M. Xie. "Optimality and Scalability Study of Existing Placement Algorithms." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **23**(4):537–549, 2004, DOI:10.1109/TCAD.2004.825870.

[CDD19] Alexander Cowtan, Silas Dilkes, Ross Duncan, Alexandre Krajenbrink, Will Simmons, and Seyon Sivarajah. "On the Qubit Routing Problem." In *14th Conference on the Theory of Quantum Computation, Communication and Cryptography*, 2019, DOI:10.4230/LIPIcs.TQC.2019.5.

[CDK04] Steven A. Cuccaro, Thomas G. Draper, Samuel A. Kutin, and David Petrie Moulton. "A new quantum ripple-carry addition circuit." arXiv:quant-ph/0410184, 2004, DOI:10.48550/arXiv.quant-ph/0410184.

[CDL11] Adán Cabello, Lars Eirik Danielsen, Antonio J. López-Tarrida, and José R. Portillo. "Optimal preparation of graph states." *Physical Review A*, **83**:042314, 2011, DOI:10.1103/PhysRevA.83.042314.

[CHH14] Parinya Chalermsook, Sandy Heydrich, Eugenia Holm, and Andreas Karrenbauer. "Nearly Tight Approximability Results for Minimum Biclique Cover and Partition." In *Algorithms - European Symposium on Algorithms 2014*, pp. 235–246, 2014, DOI:10.1007/978-3-662-44777-2_20.

[CHS93] J. Cong, M. Hossain, and N.A. Sherwani. "A provably good multilayer topological planar routing algorithm in IC layout designs." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **12**(1):70–78, 1993, DOI:10.1109/43.184844.

[CIK16] Sunil Chandran, Davis Issac, and Andreas Karrenbauer. "On the Parameterized Complexity of Biclique Cover and Partition." In *11th International Symposium on Parameterized and Exact Computation*, 2016, DOI:10.4230/LIPICS.IPEC.2016.11.

[CRS98] A Robert Calderbank, Eric M Rains, Peter M Shor, and Neil JA Sloane. "Quantum error correction via codes over GF(4)." *IEEE Transactions on Information Theory*, **44**(4):1369–1387, 1998, DOI:10.1109/ISIT.1997.613213.

[CS96] A. R. Calderbank and Peter W. Shor. "Good quantum error-correcting codes exist." *Physical Review A*, **54**:1098–1105, 1996, DOI:10.1103/PhysRevA.54.1098.

[CSU19] Andrew M. Childs, Eddie Schoute, and Cem M. Unsal. "Circuit Transformations for Quantum Architectures." In Wim van Dam and Laura Mancinska, editors, *14th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2019)*, volume 135 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 3:1–3:24, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, DOI:10.4230/LIPIcs.TQC.2019.3.

[CZY20] Christopher Chamberland, Guanyu Zhu, Theodore J. Yoder, Jared B. Hertzberg, and Andrew W. Cross. "Topological and Subsystem Codes on Low-Degree

Graphs with Flag Qubits." *Physical Review X*, **10**(1):011022, 2020, DOI:10.1103/PhysRevX.10.011022.

[dB08] Leonardo de Moura and Nikolaj Bjørner. "Z3: An Efficient SMT Solver." In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pp. 337–340, 2008, DOI:10.1007/978-3-540-78800-3_24.

[Dev21] Cirq Developers. "Cirq." Zenodo:5182845, 2021, DOI:10.5281/zenodo.5182845. See full list of authors on Github: https://github.com/quantumlib/Cirq/graphs/contributors.

[DHJ18] Yongshan Ding, Adam Holmes, Ali Javadi-Abhari, Diana Franklin, Margaret Martonosi, and Frederic T. Chong. "Magic-State Functional Units: Mapping and Scheduling Multi-Level Distillation Circuits for Fault-Tolerant Quantum Architectures." In *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-51, pp. 828–840. IEEE Press, 2018, DOI:10.1109/MICRO.2018.00072.

[Die82] D. Dieks. "Communication by EPR devices." *Physics Letters A*, **92**(6):271–272, 1982, DOI:https://doi.org/10.1016/0375-9601(82)90084-6.

[DKL02] Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. "Topological quantum memory." *Journal of Mathematical Physics*, **43**(9):4452–4505, 2002, DOI:10.1063/1.1499754.

[DMB23] Alexander M. Dalzell, Sam McArdle, Mario Berta, Przemyslaw Bienias, Chi-Fang Chen, András Gilyén, Connor T. Hann, Michael J. Kastoryano, Emil T. Khabiboulline, Aleksander Kubica, Grant Salton, Samson Wang, and Fernando G. S. L. Brandão. "Quantum algorithms: A survey of applications and end-to-end complexities." arXiv:2310.03011, 2023, DOI:10.48550/arXiv.2310.03011.

[EBK23] Simon J. Evered, Dolev Bluvstein, Marcin Kalinowski, Sepehr Ebadi, Tom Manovitz, Hengyun Zhou, Sophie H. Li, Alexandra A. Geim, Tout T. Wang, Nishad Maskara, Harry Levine, Giulia Semeghini, Markus Greiner, Vladan Vuletić, and Mikhail D. Lukin. "High-fidelity parallel entangling gates on a neutral-atom quantum computer." *Nature*, **622**(7982):268–272, 2023, DOI:10.1038/s41586-023-06481-y.

[EKC22] S. Ebadi, A. Keesling, M. Cain, T. T. Wang, H. Levine, D. Bluvstein, G. Semeghini, A. Omran, J.-G. Liu, R. Samajdar, X.-Z. Luo, B. Nash, X. Gao, B. Barak, E. Farhi, S. Sachdev, N. Gemelke, L. Zhou, S. Choi, H. Pichler, S.-T. Wang, M. Greiner, V. Vuletic, and M. D. Lukin. "Quantum optimization of maximum independent set using Rydberg atom arrays." *Science*, **376**(6598):1209–1215, 2022, DOI:10.1126/science.abo6587.

[FD13] Austin G. Fowler and Simon J. Devitt. "A bridge to lower overhead quantum computation." arXiv:1209.0510, 2013, DOI:10.48550/arXiv.1209.0510.

[FG19] Austin G. Fowler and Craig Gidney. "Low overhead quantum computation using lattice surgery." arXiv:1808.06709, 2019, DOI:10.48550/arXiv.1808.06709.

[FGG14] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. "A Quantum Approximate Optimization Algorithm." arXiv:1411.4028, 2014, DOI:10.48550/arXiv.1411.4028.

[FMM12] Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland. "Surface codes: Towards practical large-scale quantum computation." *Physical Review A*, **86**:032324, 2012, DOI:10.1103/PhysRevA.86.032324.

[FND20] B. Foxen, C. Neill, A. Dunsworth, P. Roushan, B. Chiaro, A. Megrant, J. Kelly, Zijun Chen, K. Satzinger, R. Barends, F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, S. Boixo, D. Buell, B. Burkett, Yu Chen, R. Collins, E. Farhi, A. Fowler, C. Gidney, M. Giustina, R. Graff, M. Harrigan, T. Huang, S. V. Isakov, E. Jeffrey, Z. Jiang, D. Kafri, K. Kechedzhi, P. Klimov, A. Korotkov, F. Kostritsa, D. Landhuis, E. Lucero, J. McClean, M. McEwen, X. Mi, M. Mohseni, J.Y. Mutus, O. Naaman, M. Neeley, M. Niu, A. Petukhov, C. Quintana, N. Rubin, D. Sank, V. Smelyanskiy, A. Vainsencher, T.C. White, Z. Yao, P. Yeh, A. Zalcman, H. Neven, J.M. Martinis, and Google AI Quantum. "Demonstrating a Continuous Set of Two-Qubit Gates for Near-Term Quantum Algorithms." *Physical Review Letters*, **125**(12):120504, 2020, DOI:10.1103/PhysRevLett.125.120504.

[FSG09] Austin G. Fowler, Ashley M. Stephens, and Peter Groszkowski. "High-threshold universal quantum computation on the surface code." *Physical Review A*, **80**:052312, 2009, DOI:10.1103/PhysRevA.80.052312.

[GE21] Craig Gidney and Martin Ekerå. "How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits." *Quantum*, **5**:433, 2021, DOI:10.22331/q-2021-04-15-433.

[GF19a] Craig Gidney and Austin G. Fowler. "Efficient magic state factories with a catalyzed $|CCZ\rangle$ to $2|T\rangle$ transformation." *Quantum*, **3**:135, 2019, DOI:10.22331/q-2019-04-30-135.

[GF19b] Craig Gidney and Austin G. Fowler. "Flexible layout of surface code computations using AutoCCZ states." arXiv:1905.08916, 2019, DOI:10.48550/arXiv.1905.08916.

[Gid21] Craig Gidney. "Stim: a fast stabilizer circuit simulator." *Quantum*, **5**:497, 2021, DOI:10.22331/q-2021-07-06-497.

[Gid24] Craig Gidney. "Inplace Access to the Surface Code Y Basis." *Quantum*, **8**:1310, 2024, DOI:10.22331/q-2024-04-08-1310.

[GJE20] Pranav Gokhale, Ali Javadi-Abhari, Nathan Earnest, Yunong Shi, and Frederic T. Chong. "Optimized Quantum Compilation for Near-Term Algorithms with OpenPulse." In *Proceedings of the 53rd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '20, pp. 186–200, Athens, Greece, 2020. IEEE, DOI:10.1109/MICRO50266.2020.00027.

[Got97] Daniel Gottesman. *Stabilizer codes and quantum error correction*. PhD thesis, California Institute of Technology, 1997, DOI:10.48550/arXiv.quant-ph/9705052.

[GSS22] T. M. Graham, Y. Song, J. Scott, C. Poole, L. Phuttitarn, K. Jooya, P. Eichler, X. Jiang, A. Marra, B. Grinkemeyer, M. Kwon, M. Ebert, J. Cherek, M. T. Lichtman, M. Gillette, J. Gilbert, D. Bowman, T. Ballance, C. Campbell, E. D. Dahl, O. Crawford, N. S. Blunt, B. Rogers, T. Noel, and M. Saffman. "Multi-qubit entanglement and algorithms on a neutral-atom quantum computer." *Nature*, **604**(7906):457–462, 2022, DOI:10.1038/s41586-022-04603-6.

[HCJ21] Fei Hua, Yanhao Chen, Yuwei Jin, Chi Zhang, Ari Hayes, Youtao Zhang, and Eddy Z. Zhang. "AutoBraid: A Framework for Enabling Efficient Surface Code Communication in Quantum Computing." In *54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, pp. 925–936, New York, NY, USA, 2021. Association for Computing Machinery, DOI:10.1145/3466752.3480072.

[HDE06] M. Hein, W. Dür, J. Eisert, R. Raussendorf, M. Van den Nest, and H.-J. Briegel. "Entanglement in graph states and its applications." In *Quantum Computers, Algorithms and Chaos*, volume 162 of *Proceedings of the Enrico Fermi International School of Physics*, pp. 115–218, Varenna, Italy, 2006. IOP Press, DOI:10.3254/978-1-61499-018-5-115.

[HFD12] Dominic Horsman, Austin G Fowler, Simon Devitt, and Rodney Van Meter. "Surface code quantum computing by lattice surgery." *New Journal of Physics*, **14**(12):123011, 2012, DOI:10.1088/1367-2630/14/12/123011.

[HLT21] Chen-Hao Hsu, Wan-Hsuan Lin, Wei-Hsiang Tseng, and Yao-Wen Chang. "A Bridge-based Compression Algorithm for Topological Quantum Circuits." In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 457–462, 2021, DOI:10.1109/DAC18074.2021.9586322.

[HND17] Daniel Herr, Franco Nori, and Simon J Devitt. "Lattice surgery translation for quantum computation." *New Journal of Physics*, **19**(1):013034, 2017, DOI:10.1088/1367-2630/aa5709.

[HNY11] Yuichi Hirata, Masaki Nakanishi, Shigeru Yamashita, and Yasuhiko Nakashima. "An efficient conversion of quantum circuits to a linear nearest neighbor architecture." *Quantum Info. Comput.*, **11**(1):142–166, 2011, DOI:10.5555/2011383.2011393.

[HSN21] Matthew P. Harrigan, Kevin J. Sung, Matthew Neeley, Kevin J. Satzinger, Frank Arute, Kunal Arya, Juan Atalaya, Joseph C. Bardin, Rami Barends, Sergio Boixo, Michael Broughton, Bob B. Buckley, David A. Buell, Brian Burkett, Nicholas Bushnell, Yu Chen, Zijun Chen, Ben Chiaro, Roberto Collins, William Courtney, Sean Demura, Andrew Dunsworth, Daniel Eppens, Austin Fowler, Brooks Foxen, Craig Gidney, Marissa Giustina, Rob Graff, Steve Habegger, Alan Ho, Sabrina Hong, Trent Huang, L. B. Ioffe, Sergei V. Isakov, Evan Jeffrey, Zhang Jiang, Cody Jones, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Seon Kim, Paul V. Klimov, Alexander N. Korotkov, Fedor Kostritsa, David Landhuis, Pavel Laptev, Mike Lindmark, Martin Leib, Orion Martin, John M. Martinis, Jarrod R. McClean, Matt McEwen, Anthony Megrant, Xiao Mi, Masoud Mohseni, Wojciech Mruczkiewicz, Josh Mutus, Ofer Naaman, Charles Neill, Florian Neukart, Murphy Yuezhen Niu, Thomas E. O'Brien, Bryan O'Gorman, Eric Ostby, Andre Petukhov, Harald Putterman, Chris Quintana, Pedram Roushan, Nicholas C. Rubin, Daniel Sank, Andrea Skolik, Vadim Smelyanskiy, Doug Strain, Michael Streif, Marco Szalay, Amit Vainsencher, Theodore White, Z. Jamie Yao, Ping Yeh, Adam Zalcman, Leo Zhou, Hartmut Neven, Dave Bacon, Erik Lucero, Edward Farhi, and Ryan Babbush. "Quantum approximate optimization of non-planar graph problems on a planar superconducting processor." *Nature Physics*, **17**(3):332–336, 2021, DOI:10.1038/s41567-020-01105-y.

[HSS08] Aric Hagberg, Pieter Swart, and Daniel S Chult. "Exploring network structure, dynamics, and function using NetworkX." Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008, DOI:10.25080/tcwv9851.

[HSS19] Demian Hespe, Christian Schulz, and Darren Strash. "Scalable Kernelization for Maximum Independent Sets." *ACM Journal of Experimental Algorithmics*, **24**(1), 2019, DOI:10.1145/3355502.

[Hun07] J. D. Hunter. "Matplotlib: A 2D graphics environment." *Computing in Science & Engineering*, **9**(3):90–95, 2007, DOI:10.1109/MCSE.2007.55.

[HWO19] Stuart Hadfield, Zhihui Wang, Bryan O'Gorman, Eleanor Rieffel, Davide Venturelli, and Rupak Biswas. "From the Quantum Approximate Optimization Algorithm to a Quantum Alternating Operator Ansatz." *Algorithms*, **12**(2):34, 2019, DOI:10.3390/a12020034.

[IMM18] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. "PySAT: A Python Toolkit for Prototyping with SAT Oracles." In *SAT*, pp. 428–437, 2018, DOI:10.1007/978-3-319-94144-8_26.

[IMM22] Raban Iten, Romain Moyard, Tony Metger, David Sutter, and Stefan Woerner. "Exact and Practical Pattern Matching for Quantum Circuit Optimization." *ACM Transactions on Quantum Computing*, **3**(1), 2022, DOI:10.1145/3498325.

[IRI19] Toshinari Itoko, Rudy Raymond, Takashi Imamichi, Atsushi Matsuo, and Andrew W. Cross. "Quantum Circuit Compilers Using Gate Commutation Rules." In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, ASPDAC '19, pp. 191–196, New York, NY, USA, 2019. Association for Computing Machinery, DOI:10.1145/3287624.3287701.

[JGH17] Ali Javadi-Abhari, Pranav Gokhale, Adam Holmes, Diana Franklin, Kenneth R. Brown, Margaret Martonosi, and Frederic T. Chong. "Optimized Surface Code Communication in Superconducting Quantum Computers." In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-50 '17, pp. 692–705, New York, NY, USA, 2017. Association for Computing Machinery, DOI:10.1145/3123939.3123949.

[JJB21] Petar Jurcevic, Ali Javadi-Abhari, Lev S Bishop, Isaac Lauer, Daniela F Bogorin, Markus Brink, Lauren Capelluto, Oktay Günlük, Toshinari Itoko, Naoki Kanazawa, Abhinav Kandala, George A Keefe, Kevin Krsulich, William Landers, Eric P Lewandowski, Douglas T McClure, Giacomo Nannicini, Adinath Narasgond, Hasan M Nayfeh, Emily Pritchett, Mary Beth Rothwell, Srikanth Srinivasan, Neereja Sundaresan, Cindy Wang, Ken X Wei, Christopher J Wood, Jeng-Bang Yau, Eric J Zhang, Oliver E Dial, Jerry M Chow, and Jay M Gambetta. "Demonstration of quantum volume 64 on a superconducting quantum computing system." *Quantum Science and Technology*, **6**(2):025020, 2021, DOI:10.1088/2058-9565/abe519.

[JR93] Tao Jiang and B. Ravikumar. "Minimal NFA Problems are Hard." *SIAM Journal on Computing*, **22**(6):1117–1141, 1993, DOI:10.1137/0222067.

[Kar72] Richard M. Karp. "Reducibility among Combinatorial Problems." In Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pp. 85–103. Springer, Boston, MA, 1972, DOI:10.1007/978-1-4684-2001-2_9.

[KDS16] A. Kole, K. Datta, and I. Sengupta. "A Heuristic for Linear Nearest Neighbor Realization of Quantum Circuits by SWAP Gate Insertion Using N-Gate Lookahead." *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, **6**(1):62–72, 2016, DOI:10.1109/JETCAS.2016.2528720.

[KDS18] A. Kole, K. Datta, and I. Sengupta. "A New Heuristic for N-Dimensional Nearest Neighbor Realization of a Quantum Circuit." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **37**(1):182–192, 2018, DOI:10.1109/TCAD.2017.2693284.

[KHD19] A. Kole, S. Hillmich, K. Datta, R. Wille, and I. Sengupta. "Improved Mapping of Quantum Circuits to IBM QX Architectures." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2019, DOI:10.1109/TCAD.2019.2962753.

[Kit03] A.Yu. Kitaev. "Fault-tolerant quantum computation by anyons." *Annals of Physics*, **303**(1):2–30, 2003, DOI:https://doi.org/10.1016/S0003-4916(02)00018-0.

[KLZ98] Emanuel Knill, Raymond Laflamme, and Wojciech H. Zurek. "Resilient Quantum Computation." *Science*, **279**(5349):342–345, 1998, DOI:10.1126/science.279.5349.342.

[KMW02] D. Kielpinski, C. Monroe, and D. J. Wineland. "Architecture for a large-scale ion-trap quantum computer." *Nature*, **417**(6890):709–711, 2002, DOI:10.1038/nature00784.

[KMW18] Ian D. Kivlichan, Jarrod McClean, Nathan Wiebe, Craig Gidney, Alán Aspuru-Guzik, Garnet Kin-Lic Chan, and Ryan Babbush. "Quantum Simulation of Electronic Structure with Linear Depth and Connectivity." *Physical Review Letters*, **120**(11):110501, 2018, DOI:10.1103/PhysRevLett.120.110501.

[KN97] Eyal Kushilevitz and Noam Nisan. *Communication Complexity*, chapter 1.1. Cambridge University Press, 1997, DOI:10.1017/CBO9780511574948.

[LBM23] Sitong Liu, Naphan Benchasattabuse, Darcy QC Morgan, Michal Hajdušek, Simon J. Devitt, and Rodney Van Meter. "A Substrate Scheduler for Compiling Arbitrary Fault-Tolerant Graph States." In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 01, pp. 870–880, 2023, DOI:10.1109/QCE57702.2023.00101.

[LBP16] Dawei Lu, Aharon Brodutch, Jihyun Park, Hemant Katiyar, Tomas Jochym-O'Connor, and Raymond Laflamme. "NMR Quantum Information Processing." In Takeji Takui, Lawrence Berliner, and Graeme Hanson, editors, *Electron Spin Resonance (ESR) Based Quantum Computing*, pp. 193–226. Springer New York, New York, NY, 2016, DOI:10.1007/978-1-4939-3658-8_7.

[LDX19] Gushu Li, Yufei Ding, and Yuan Xie. "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, Providence, RI, USA, 2019. DOI:10.1145/3297858.3304023.

[Lit19a] Daniel Litinski. "A Game of Surface Codes: Large-Scale Quantum Computing with Lattice Surgery." *Quantum*, **3**:128, 2019, DOI:10.22331/q-2019-03-05-128.

[Lit19b] Daniel Litinski. "Magic State Distillation: Not as Costly as You Think." *Quantum*, **3**:205, 2019, DOI:10.22331/q-2019-12-02-205.

[LKS19] Harry Levine, Alexander Keesling, Giulia Semeghini, Ahmed Omran, Tout T. Wang, Sepehr Ebadi, Hannes Bernien, Markus Greiner, Vladan Vuletić, Hannes Pichler, and Mikhail D. Lukin. "Parallel implementation of high-fidelity multiqubit gates with neutral atoms." *Physical Review Letters*, **123**(17):170503, 2019, DOI:10.1103/PhysRevLett.123.170503.

[LKT23] Wan-Hsuan Lin, Jason Kimko, Bochen Tan, Nikolaj Bjørner, and Jason Cong. "Scalable Optimal Layout Synthesis for NISQ Quantum Processors." In *Proceedings of the 60th ACM/IEEE Design Automation Conference (DAC)*, 2023, DOI:10.1109/DAC56929.2023.10247760.

[LN22] Daniel Litinski and Naomi Nickerson. "Active volume: An architecture for efficient fault-tolerant quantum computers with limited non-local connections." arXiv:2211.15465, 2022, DOI:10.48550/arXiv.2211.15465.

[LSK22] Ang Li, Samuel Stein, Sriram Krishnamoorthy, and James Ang. "QASMBench: A Low-Level Quantum Benchmark Suite for NISQ Evaluation and Simulation." *ACM Transactions on Quantum Computing*, 2022, DOI:10.1145/3550488.

[LTN22] Wan-Hsuan Lin, Bochen Tan, Murphy Yuezhen Niu, Jason Kimko, and Jason Cong. "Domain-Specific Quantum Architecture Optimization." *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, **12**(3):624–637, 2022, DOI:10.1109/JETCAS.2022.3202870.

[LWD15] A. Lye, R. Wille, and R. Drechsler. "Determining the minimal number of SWAP gates for multi-dimensional nearest neighbor quantum circuits." In *The 20th Asia and South Pacific Design Automation Conference*, pp. 178–183, 2015, DOI:10.1109/ASPDAC.2015.7059001.

[LYL18] Yibo Lin, Bei Yu, Meng Li, and David Z. Pan. "Layout Synthesis for Topological Quantum Circuits With 1-D and 2-D Architectures." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **37**(8):1574–1587, 2018, DOI:10.1109/TCAD.2017.2760511.

[LZC23] Yongshang Li, Yu Zhang, Mingyu Chen, Xiangyang Li, and Peng Xu. "Timing-Aware Qubit Mapping and Gate Scheduling Adapted to Neutral Atom Quantum Computing." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **42**(11):3768–3780, 2023, DOI:10.1109/TCAD.2023.3261244.

[MBA23] S. A. Moses, C. H. Baldwin, M. S. Allman, R. Ancona, L. Ascarrunz, C. Barnes, J. Bartolotta, B. Bjork, P. Blanchard, M. Bohn, J. G. Bohnet, N. C. Brown, N. Q. Burdick, W. C. Burton, S. L. Campbell, J. P. Campora, C. Carron, J. Chambers,

J. W. Chan, Y. H. Chen, A. Chernoguzov, E. Chertkov, J. Colina, J. P. Curtis, R. Daniel, M. DeCross, D. Deen, C. Delaney, J. M. Dreiling, C. T. Ertsgaard, J. Esposito, B. Estey, M. Fabrikant, C. Figgatt, C. Foltz, M. Foss-Feig, D. Francois, J. P. Gaebler, T. M. Gatterman, C. N. Gilbreth, J. Giles, E. Glynn, A. Hall, A. M. Hankin, A. Hansen, D. Hayes, B. Higashi, I. M. Hoffman, B. Horning, J. J. Hout, R. Jacobs, J. Johansen, L. Jones, J. Karcz, T. Klein, P. Lauria, P. Lee, D. Liefer, S. T. Lu, D. Lucchetti, C. Lytle, A. Malm, M. Matheny, B. Mathewson, K. Mayer, D. B. Miller, M. Mills, B. Neyenhuis, L. Nugent, S. Olson, J. Parks, G. N. Price, Z. Price, M. Pugh, A. Ransford, A. P. Reed, C. Roman, M. Rowe, C. Ryan-Anderson, S. Sanders, J. Sedlacek, P. Shevchuk, P. Siegfried, T. Skripka, B. Spaun, R. T. Sprenkle, R. P. Stutz, M. Swallows, R. I. Tobey, A. Tran, T. Tran, E. Vogt, C. Volin, J. Walker, A. M. Zolot, and J. M. Pino. "A Race-Track Trapped-Ion Quantum Processor." *Physical Review X*, **13**:041052, 2023, DOI:10.1103/PhysRevX.13.041052.

[MBG23] Matt McEwen, Dave Bacon, and Craig Gidney. "Relaxing Hardware Requirements for Surface Code Circuits using Time-dynamics." *Quantum*, **7**:1172, 2023, DOI:10.22331/q-2023-11-07-1172.

[MBJ19] Prakash Murali, Jonathan M. Baker, Ali Javadi-Abhari, Frederic T. Chong, and Margaret Martonosi. "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*, ASPLOS '19, pp. 1015–1029, New York, NY, USA, 2019. Association for Computing Machinery, DOI:10.1145/3297858.3304075.

[MC80] Carver Mead and Lynn Conway. *Introduction to VLSI Systems*. Addison-Wesley, USA, 1st edition, 1980.

[MFM08] Dmitri Maslov, Sean M. Falconer, and Michele Mosca. "Quantum Circuit Placement." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **27**(4):752–763, 2008, DOI:10.1109/TCAD.2008.917562.

[MG92] J. Misra and David Gries. "A constructive proof of Vizing's theorem." *Information Processing Letters*, **41**(3):131–133, 1992, DOI:10.1016/0020-0190(92)90041-S.

[MHP23] David C. McKay, Ian Hincks, Emily J. Pritchett, Malcolm Carroll, Luke C. G. Govia, and Seth T. Merkel. "Benchmarking Quantum Processor Performance at Scale." arXiv:2311.05933, 2023, DOI:10.48550/arXiv.2311.05933.

[MLM19] Prakash Murali, Norbert Matthias Linke, Margaret Martonosi, Ali Javadi Abhari, Nhung Hong Nguyen, and Cinthia Huerta Alderete. "Full-Stack, Real-System Quantum Computer Studies: Architectural Comparisons and Design

Insights." In *Proceedings of the 46th International Symposium on Computer Architecture*, ISCA '19, pp. 527–540, Phoenix, Arizona, 2019. ACM Press, DOI:10.1145/3307650.3322273.

[MVM23] A. Morvan, B. Villalonga, X. Mi, S. Mandrà, A. Bengtsson, P. V. Klimov, Z. Chen, S. Hong, C. Erickson, I. K. Drozdov, J. Chau, G. Laun, R. Movassagh, A. Asfaw, L. T. A. N. Brandão, R. Peralta, D. Abanin, R. Acharya, R. Allen, T. I. Andersen, K. Anderson, M. Ansmann, F. Arute, K. Arya, J. Atalaya, J. C. Bardin, A. Bilmes, G. Bortoli, A. Bourassa, J. Bovaird, L. Brill, M. Broughton, B. B. Buckley, D. A. Buell, T. Burger, B. Burkett, N. Bushnell, J. Campero, H. S. Chang, B. Chiaro, D. Chik, C. Chou, J. Cogan, R. Collins, P. Conner, W. Courtney, A. L. Crook, B. Curtin, D. M. Debroy, A. Del Toro Barba, S. Demura, A. Di Paolo, A. Dunsworth, L. Faoro, E. Farhi, R. Fatemi, V. S. Ferreira, L. Flores Burgos, E. Forati, A. G. Fowler, B. Foxen, G. Garcia, E. Genois, W. Giang, C. Gidney, D. Gilboa, M. Giustina, R. Gosula, A. Grajales Dau, J. A. Gross, S. Habegger, M. C. Hamilton, M. Hansen, M. P. Harrigan, S. D. Harrington, P. Heu, M. R. Hoffmann, T. Huang, A. Huff, W. J. Huggins, L. B. Ioffe, S. V. Isakov, J. Iveland, E. Jeffrey, Z. Jiang, C. Jones, P. Juhas, D. Kafri, T. Khattar, M. Khezri, M. Kieferová, S. Kim, A. Kitaev, A. R. Klots, A. N. Korotkov, F. Kostritsa, J. M. Kreikebaum, D. Landhuis, P. Laptev, K. M. Lau, L. Laws, J. Lee, K. W. Lee, Y. D. Lensky, B. J. Lester, A. T. Lill, W. Liu, W. P. Livingston, A. Locharla, F. D. Malone, O. Martin, S. Martin, J. R. McClean, M. McEwen, K. C. Miao, A. Mieszala, S. Montazeri, W. Mruczkiewicz, O. Naaman, M. Neeley, C. Neill, A. Nersisyan, M. Newman, J. H. Ng, A. Nguyen, M. Nguyen, M. Yuezhen Niu, T. E. O'Brien, S. Omonije, A. Opremcak, A. Petukhov, R. Potter, L. P. Pryadko, C. Quintana, D. M. Rhodes, E. Rosenberg, C. Rocque, P. Roushan, N. C. Rubin, N. Saei, D. Sank, K. Sankaragomathi, K. J. Satzinger, H. F. Schurkus, C. Schuster, M. J. Shearn, A. Shorter, N. Shutty, V. Shvarts, V. Sivak, J. Skruzny, W. C. Smith, R. D. Somma, G. Sterling, D. Strain, M. Szalay, D. Thor, A. Torres, G. Vidal, C. Vollgraff Heidweiller, T. White, B. W. K. Woo, C. Xing, Z. J. Yao, P. Yeh, J. Yoo, G. Young, A. Zalcman, Y. Zhang, N. Zhu, N. Zobrist, E. G. Rieffel, R. Biswas, R. Babbush, D. Bacon, J. Hilton, E. Lucero, H. Neven, A. Megrant, J. Kelly, I. Aleiner, V. Smelyanskiy, K. Kechedzhi, Y. Chen, and S. Boixo. "Phase transition in Random Circuit Sampling." arXiv:2304.11119, 2023, DOI:10.48550/arXiv.2304.11119.

[NBG22] Giacomo Nannicini, Lev S. Bishop, Oktay Günlük, and Petar Jurcevic. "Optimal Qubit Assignment and Routing via Integer Programming." *ACM Transactions on Quantum Computing*, 2022, DOI:10.1145/3544563.

[NC10] Michael A Nielsen and Isaac L Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, Cambridge, United Kingdom, 2010, DOI:10.1017/CBO9780511976667.

[NPW23] Natalia Nottingham, Michael A. Perlin, Ryan White, Hannes Bernien, Frederic T. Chong, and Jonathan M. Baker. "Decomposing and Routing Quantum Circuits Under Constraints for Neutral Atom Architectures." arXiv:2307.14996, 2023, DOI:10.48550/arXiv.2307.14996.

[NRS18] Yunseong Nam, Neil J. Ross, Yuan Su, Andrew M. Childs, and Dmitri Maslov. "Automated Optimization of Large Quantum Circuits with Continuous Parameters." *npj Quantum Information*, **4**(1):23, 2018, DOI:10.1038/s41534-018-0072-4.

[Pal15] Alexandru Paler. *Design Methods for Reliable Quantum Circuits*. PhD thesis, Universität Passau, 2015.

[PCS20] Eric C. Peterson, Gavin E. Crooks, and Robert S. Smith. "Fixed-Depth Two-Qubit Circuits and the Monodromy Polytope." *Quantum*, **4**:247, 2020, DOI:10.22331/q-2020-03-26-247.

[PDF16] Alexandru Paler, Simon J Devitt, and Austin G Fowler. "Synthesis of arbitrary quantum circuits to topological assembly." *Scientific reports*, **6**(1):30600, 2016, DOI:10.1038/srep30600.

[PF20] Alexandru Paler and Austin G. Fowler. "OpenSurgery for Topological Assemblies." In *2020 IEEE Globecom Workshops*, 2020, DOI:10.1109/GCWkshps50303.2020.9367489.

[PM21] Sébastien Pezzagna and Jan Meijer. "Quantum computer based on color centers in diamond." *Applied Physics Reviews*, **8**(1):011308, 2021, DOI:10.1063/5.0007444.

[PST22] Tirthak Patel, Daniel Silver, and Devesh Tiwari. "Geyser: A Compilation Framework for Quantum Computing with Neutral Atoms." In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ISCA '22, pp. 383–395, New York, NY, USA, 2022. Association for Computing Machinery, DOI:10.1145/3470496.3527428.

[RHG07] R Raussendorf, J Harrington, and K Goyal. "Topological fault-tolerance in cluster state quantum computation." *New Journal of Physics*, **9**(6):199, 2007, DOI:10.1088/1367-2630/9/6/199.

[SAP22] Kevin Singh, Shraddha Anand, Andrew Pocklington, Jordan T. Kemp, and Hannes Bernien. "Dual-Element, Two-Dimensional Atom Array with Continuous-Mode Operation." *Physical Review X*, **12**(1):011040, 2022, DOI:10.1103/PhysRevX.12.011040.

[SB80] Sartaj Sahni and Atul Bhatt. "The Complexity of Design Automation Problems." In *Proceedings of the 17th Design Automation Conference*, DAC '80, pp. 402–411, New York, NY, USA, 1980. Association for Computing Machinery, DOI:10.1145/800139.804562.

[SC22] Noah Shutty and Christopher Chamberland. "Decoding Merged Color-Surface Codes and Finding Fault-Tolerant Clifford Circuits Using Solvers for Satisfiability Modulo Theories." *Physical Review Applied*, **18**(1):014072, 2022, DOI:10.1103/PhysRevApplied.18.014072.

[SDC20] Seyon Sivarajah, Silas Dilkes, Alexander Cowtan, Will Simmons, Alec Edgington, and Ross Duncan. "t|ket⟩: A Retargetable Compiler for NISQ Devices." *Quantum Science and Technology*, **6**(1):014003, 2020, DOI:10.1088/2058-9565/ab8e92.

[Sem07] Semiconductor Research Corporation. "'Huge opportunity' in IC design optimization gained by Semiconductor Research Corporation, National Science Foundation: CAD innovation could save industry billions." SRC Press Release, 2007, https://www.src.org/newsroom/press-release/2007/41/.

[Sho94] P.W. Shor. "Algorithms for Quantum Computation: Discrete Logarithms and Factoring." In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pp. 124–134, Santa Fe, NM, USA, 1994. IEEE Comput. Soc. Press, DOI:10.1109/SFCS.1994.365700.

[Sho99] Peter W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer." *SIAM Review*, **41**(2):303–332, 1999, DOI:10.1137/S0036144598347011.

[SLG19] Yunong Shi, Nelson Leung, Pranav Gokhale, Zane Rossi, David I. Schuster, Henry Hoffmann, and Frederic T. Chong. "Optimized Compilation of Aggregated Instructions for Realistic Quantum Computers." In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 1031–1044, New York, NY, USA, 2019. DOI:10.1145/3297858.3304018.

[SLT24] Atefeh Sohrabizadeh, Wan-Hsuan Lin, Daniel Bochen Tan, Madelyn Cain, Sheng-Tao Wang, Mikhail D. Lukin, and Jason Cong. "GNN-Based Performance Prediction of Quantum Optimization of Maximum Independent Set." In *The 43rd International Conference on Computer-Aided Design*, 2024.

[SPK23] Ludwig Schmid, Sunghye Park, Seokhyeong Kang, and Robert Wille. "Hybrid Circuit Mapping: Leveraging the Full Spectrum of Computational Capabilities of Neutral Atom Quantum Computers." arXiv:2311.14164, 2023, DOI:10.48550/arXiv.2311.14164.

[SPM03] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes. "Synthesis of reversible logic circuits." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **22**(6):710–722, 2003, DOI:10.1109/TCAD.2003.811448.

[SPS20] R. S. Smith, E. C. Peterson, M. G. Skilbeck, and E. J. Davis. "An open-source, industrial-strength optimizing compiler for quantum programs." *Quantum Science and Technology*, **5**(4):044001, 2020, DOI:10.1088/2058-9565/ab9acb.

[SRA11] Thomas Szkopek, Vwani P. Roychowdhury, Dimitri A. Antoniadis, and John N. Damoulakis. "Physical Fault Tolerance of Nanoelectronics." *Physical Review Letters*, **106**:176801, 2011, DOI:10.1103/PhysRevLett.106.176801.

[SS21] Dominik Schreiber and Peter Sanders. *Scalable SAT Solving in the Cloud*, pp. 518–534. Springer, Cham, 2021, DOI:10.1007/978-3-030-80223-3_35.

[SSC18] Marcos Yukio Siraichi, Vinícius Fernandes dos Santos, Sylvain Collange, and Fernando Magno Quintao Pereira. "Qubit Allocation." In *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, CGO '18, pp. 113–125, Vienna, Austria, 2018. ACM Press, DOI:10.1145/3168822.

[SSC19] Marcos Yukio Siraichi, Vinícius Fernandes dos Santos, Caroline Collange, and Fernando Magno Quintão Pereira. "Qubit Allocation as a Combination of Subgraph Isomorphism and Token Swapping." *Proc. ACM Program. Lang.*, **3**(OOPSLA), 2019, DOI:10.1145/3360546.

[SSP13] Alireza Shafaei, Mehdi Saeedi, and Massoud Pedram. "Optimization of Quantum Circuits for Interaction Distance in Linear Nearest Neighbor Architectures." In *Proceedings of the 50th Annual Design Automation Conference*, DAC '13, New York, NY, USA, 2013. Association for Computing Machinery, DOI:10.1145/2463209.2488785.

[SSP14] Alireza Shafaei, Mehdi Saeedi, and Massoud Pedram. "Qubit Placement to Minimize Communication Overhead in 2D Quantum Architectures." In *Proceedings of the 19th Asia and South Pacific Design Automation Conference*, ASP-DAC '14, pp. 495–500, Singapore, 2014. IEEE, DOI:10.1109/ASPDAC.2014.6742940.

[Ste96] Andrew Steane. "Multiple-particle interference and quantum error correction." *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, **452**(1954):2551–2577, 1996, DOI:10.1098/rspa.1996.0136.

[TBL22] Bochen Tan, Dolev Bluvstein, Mikhail D. Lukin, and Jason Cong. "Qubit Mapping for Reconfigurable Atom Arrays." In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '22, San Diego, USA, 2022. DOI:10.1145/3508352.3549331.

[TBL24] Daniel Bochen Tan, Dolev Bluvstein, D. Mikhail Lukin, and Jason Cong. "Compiling quantum circuits for dynamically field-programmable neutral atoms array processors." *Quantum*, **8**:1281, 2024, DOI:10.22331/q-2024-03-14-1281.

[TC20] Bochen Tan and Jason Cong. "Optimal Layout Synthesis for Quantum Computing." In *Proceedings of the 39th IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '20, Virtual Event, USA, 2020. Association for Computing Machinery, DOI:10.1145/3400302.3415620.

[TC21a] Bochen Tan and Jason Cong. "Optimal Qubit Mapping with Simultaneous Gate Absorption." In *Proceedings of the 40th IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '21, Munich, Germany, 2021. Association for Computing Machinery, DOI:10.1109/ICCAD51958.2021.9643554.

[TC21b] Bochen Tan and Jason Cong. "Optimality Study of Existing Quantum Computing Layout Synthesis Tools." *IEEE Transactions on Computers*, **70**(9):1363–1373, 2021, DOI:10.1109/TC.2020.3009140.

[TC23] Bochen Tan and Jason Cong. "Layout Synthesis for Near-Term Quantum Computing: Gap Analysis and Optimal Solution." In Rasit O. Topaloglu, editor, *Design Automation of Quantum Computers*, pp. 25–40. Springer International Publishing, Cham, 2023, DOI:10.1007/978-3-031-15699-1_2.

[TLC24] Daniel Bochen Tan, Wan-Hsuan Lin, and Jason Cong. "Compilation for Dynamically Field-Programmable Qubit Arrays with Efficient and Provably Near-Optimal Scheduling." In *The 30th Asia and South Pacific Design Automation Conference*, 2024, DOI:10.48550/arXiv.2405.15095.

[TMN17] Swamit S. Tannu, Zachary A. Myers, Prashant J. Nair, Douglas M. Carmean, and Moinuddin K. Qureshi. "Taming the Instruction Bandwidth of Quantum Computers via Hardware-Managed Error Correction." In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-50 '17, pp. 679–691, New York, NY, USA, 2017. Association for Computing Machinery, DOI:10.1145/3123939.3123940.

[TNG24] Daniel Bochen Tan, Murphy Yuezhen Niu, and Craig Gidney. "A SAT Scalpel for Lattice Surgery: Representation and Synthesis of Subroutines for Surface-Code Fault-Tolerant Quantum Computing." In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pp. 325–339, Los Alamitos, CA, USA, 2024. IEEE Computer Society, DOI:10.1109/ISCA59077.2024.00032.

[TPC24] Daniel Bochen Tan, Shuohao Ping, and Jason Cong. "Depth-Optimal Addressing of 2D Qubit Array with 1D Controls Based on Exact Binary Matrix Factorization." In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2024, DOI:10.48550/arXiv.2401.13807.

[TQ19] Swamit S. Tannu and Moinuddin K. Qureshi. "Not All Qubits Are Created Equal: A Case for Variability-Aware Policies for NISQ-Era Quantum Computers." In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS '19*, ASPLOS '19, pp. 987–999, Providence, RI, USA, 2019. Association for Computing Machinery, DOI:10.1145/3297858.3304007.

[VDR17] Davide Venturelli, Minh Do, Eleanor Rieffel, and Jeremy Frank. "Temporal Planning for Compilation of Quantum Approximate Optimization Circuits." In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pp. 4440–4446, 2017, DOI:10.24963/ijcai.2017/620.

[VDR18] Davide Venturelli, Minh Do, Eleanor Rieffel, and Jeremy Frank. "Compiling Quantum Circuits to Realistic Hardware Architectures Using Temporal Planners." *Quantum Science and Technology*, **3**(2):025004, 2018, DOI:10.1088/2058-9565/aaa331.

[Viz65] V. G. Vizing. "Critical graphs with given chromatic class." *Metody Diskret. Analiz.*, **5**:9–17, 1965.

[VPG24] Madhav Krishnan Vijayan, Alexandru Paler, Jason Gavriel, Casey R Myers, Peter P Rohde, and Simon J Devitt. "Compilation of algorithm-specific graph states for quantum circuits." *Quantum Science and Technology*, **9**(2):025005, 2024, DOI:10.1088/2058-9565/ad1f39.

[VW04] Farrokh Vatan and Colin Williams. "Optimal quantum circuits for general two-qubit gates." *Physical Review A*, **69**(3):032315, 2004, DOI:10.1103/PhysRevA.69.032315.

[Wat16] Thomas Watson. "Nonnegative Rank vs. Binary Rank." *Chicago Journal of Theoretical Computer Science*, **22**(1), 2016, DOI:10.4086/cjtcs.2016.002.

[WBC21] Yulin Wu, Wan-Su Bao, Sirui Cao, Fusheng Chen, Ming-Cheng Chen, Xiawei Chen, Tung-Hsun Chung, Hui Deng, Yajie Du, Daojin Fan, Ming Gong, Cheng Guo, Chu Guo, Shaojun Guo, Lianchen Han, Linyin Hong, He-Liang Huang, Yong-Heng Huo, Liping Li, Na Li, Shaowei Li, Yuan Li, Futian Liang, Chun Lin, Jin Lin, Haoran Qian, Dan Qiao, Hao Rong, Hong Su, Lihua Sun, Liangyuan Wang, Shiyu Wang, Dachao Wu, Yu Xu, Kai Yan, Weifeng Yang, Yang Yang, Yangsen Ye, Jianghan Yin, Chong Ying, Jiale Yu, Chen Zha, Cha Zhang, Haibin Zhang, Kaili Zhang, Yiming Zhang, Han Zhao, Youwei Zhao, Liang Zhou, Qingling Zhu, Chao-Yang Lu, Cheng-Zhi Peng, Xiaobo Zhu, and Jian-Wei Pan. "Strong Quantum Computational Advantage Using a Superconducting Quantum Processor." *Physical Review Letters*, **127**:180501, 2021, DOI:10.1103/PhysRevLett.127.180501.

[WBZ19] Robert Wille, Lukas Burgholzer, and Alwin Zulehner. "Mapping Quantum Circuits to IBM QX Architectures Using the Minimal Number of SWAP and H Operations." In *Proceedings of the 56th Annual Design Automation Conference 2019*, Las Vegas, NV, USA, 2019. DOI:10.1145/3316781.3317859.

[Wet20] John van de Wetering. "ZX-calculus for the working quantum computer scientist." arXiv:2012.13966, 2020, DOI:10.48550/arXiv.2012.13966.

[WIP07] Mark Whitney, Nemanja Isailovic, Yatish Patel, and John Kubiatowicz. "Automated Generation of Layout and Control for Quantum Circuits." In *Proceedings of the 4th International Conference on Computing Frontiers*, CF '07, pp. 83–94, New York, NY, USA, 2007. Association for Computing Machinery, DOI:10.1145/1242531.1242546.

[WLD14] Robert Wille, Aaron Lye, and Rolf Drechsler. "Optimal SWAP Gate Insertion for Nearest Neighbor Quantum Circuits." In *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 489–494, Singapore, 2014. IEEE, DOI:10.1109/ASPDAC.2014.6742939.

[WLT24] Hanrui Wang, Pengyu Liu, Daniel Bochen Tan, Yilian Liu, Jiaqi Gu, David Z. Pan, Jason Cong, Umut A. Acar, and Song Han. "Atomique: A Quantum Compiler for Reconfigurable Neutral Atom Arrays." In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pp. 293–309, 2024, DOI:10.1109/ISCA59077.2024.00030.

[WNS23] George Watkins, Hoang Minh Nguyen, Varun Seshadri, Keelan Watkins, Steven Pearce, Hoi-Kwan Lau, and Alexandru Paler. "A High Performance Compiler for Very Large Scale Surface Code Computations." arXiv:2302.02459, 2023, DOI:10.48550/arXiv.2302.02459.

[WTC24a] Hanyu Wang, Bochen Tan, Jason Cong, and Giovanni De Micheli. "Quantum State Preparation Using an Exact CNOT Synthesis Formulation." arXiv:2401.01009, 2024, DOI:10.48550/arXiv.2401.01009.

[WTC24b] Hanyu Wang, Daniel Bochen Tan, and Jason Cong. "Quantum State Preparation Circuit Optimization Exploiting Don't Cares." In *The 43rd International Conference on Computer-Aided Design*, 2024, DOI:10.48550/arXiv.2409.01418.

[WTL24] Hanrui Wang, Bochen Tan, Pengyu Liu, Yilian Liu, Jiaqi Gu, Jason Cong, and Song Han. "Q-Pilot: Field Programmable Quantum Array Compilation with Flying Ancillas." In *The 61st Design Automation Conference (DAC)*, 2024, DOI:10.48550/arXiv.2311.16190.

[WZ82] William K Wootters and Wojciech H Zurek. "A single quantum cannot be cloned." *Nature*, **299**(5886):802–803, 1982, DOI:10.1038/299802a0.

[XBP24] Qian Xu, J. Pablo Bonilla Ataides, Christopher A. Pattison, Nithin Raveendran, Dolev Bluvstein, Jonathan Wurtz, Bane Vasić, Mikhail D. Lukin, Liang Jiang, and Hengyun Zhou. "Constant-overhead fault-tolerant quantum computation with reconfigurable atom arrays." *Nature Physics*, 2024, DOI:10.1038/s41567-024-02479-z.

[Yao79] Andrew Chi-Chih Yao. "Some complexity questions related to distributive computing." In *Proceedings of the eleventh annual ACM symposium on Theory of computing*, pp. 209–213, New York, NY, USA, 1979. DOI:10.1145/800135.804414.

[ZHQ21] Chi Zhang, Ari B. Hayes, Longfei Qiu, Yuwei Jin, Yanhao Chen, and Eddy Z. Zhang. "Time-optimal Qubit mapping." In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 360–374, Virtual USA, 2021. DOI:10.1145/3445814.3446706.

[ZLD07] Zhongyuan Zhang, Tao Li, Chris Ding, and Xiangsun Zhang. "Binary Matrix Factorization with Applications." In *Seventh IEEE International Conference on Data Mining*, pp. 391–400, 2007, DOI:10.1109/ICDM.2007.99.

[ZPW18] Alwin Zulehner, Alexandru Paler, and Robert Wille. "Efficient Mapping of Quantum Circuits to the IBM QX Architectures." In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1135–1138, Dresden, Germany, 2018. IEEE, DOI:10.23919/DATE.2018.8342181.

[ZW19] Alwin Zulehner and Robert Wille. "Compiling SU(4) quantum circuits to IBM QX architectures." In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, ASPDAC '19, pp. 185–190, Tokyo, Japan, 2019. ACM Press, DOI:10.1145/3287624.3287704.

[ZZ12] Marinka Zitnik and Blaz Zupan. "NIMFA: A Python Library for Nonnegative Matrix Factorization." *Journal of Machine Learning Research*, **13**:849–853, 2012, DOI:10.48550/arXiv.1808.01743.