Modular Component-Based Quantum Circuit Synthesis

CHAN GU KANG, Korea University, Republic of Korea
HAKJOO OH[*], Korea University, Republic of Korea

In this article, we present a novel method for synthesizing quantum circuits from user-supplied components. Given input-output state vectors and component quantum gates, our synthesizer aims to construct a quantum circuit that implements the provided functionality in terms of the supplied component gates. To achieve this, we basically use an enumerative search with pruning. To accelerate the procedure, however, we perform the search and pruning at the module level; instead of simply enumerating candidate circuits by appending component gates in sequence, we stack modules, which are groups of gate operations. With this modular approach, we can effectively reduce the search space by directing the search in a way that bridges the gap between the current circuit and the input-output specification. Evaluation on 17 benchmark problems shows that our technique is highly effective at synthesizing quantum circuits. Our method successfully synthesized 16 out of 17 benchmark circuits in 96.6 seconds on average. On the other hand, the conventional, gate-level synthesis algorithm succeeded in 10 problems with an average time of 639.1 seconds. Our algorithm increased the speed of the baseline by 20.3x for the 10 problems commonly solved by both approaches.

CCS Concepts: • **Computer systems organization → Quantum computing**; • **Software and its engineering → Automatic programming**.

Additional Key Words and Phrases: Quantum circuit synthesis, Quantum programming

ACM Reference Format:
Chan Gu Kang and Hakjoo Oh. 2023. Modular Component-Based Quantum Circuit Synthesis. *Proc. ACM Program. Lang.* 7, OOPSLA1, Article 87 (April 2023), 28 pages. https://doi.org/10.1145/3586039

1 INTRODUCTION

Quantum computers are expected to outperform classical computers in applications such as machine learning [Gilyén et al. 2019], quantum system simulation [Low and Chuang 2019], and unstructured search [Grover 1996]. Currently, implementing these algorithms takes the form of quantum circuits and is accomplished by applying quantum gates to qubit registers. In recent years, many frameworks and libraries for quantum circuit programming have become available (e.g., Cirq [Developers 2022] and Qiskit [Aleksandrowicz et al. 2019]), which has contributed to an increased interest in quantum computing and programming.

However, due to the fundamental differences between classical and quantum computers, programming quantum circuits is challenging. Quantum circuits' linear algebraic formalism is one of the primary sources of the difficulty. In quantum programming, data values are represented as vectors, and operations (quantum gates) are interpreted as matrices. These matrix operations are

[*] Corresponding author

Authors' addresses: Chan Gu Kang, changukang@korea.ac.kr, Department of Computer Science and Engineering, Korea University, 145, Anam-ro, Sungbuk-gu, Seoul, 02841, Republic of Korea; Hakjoo Oh, hakjoo_oh@korea.ac.kr, Department of Computer Science and Engineering, Korea University, 145, Anam-ro, Sungbuk-gu, Seoul, 02841, Republic of Korea.

2475-1421/2023/4-ART87
https://doi.org/10.1145/3586039

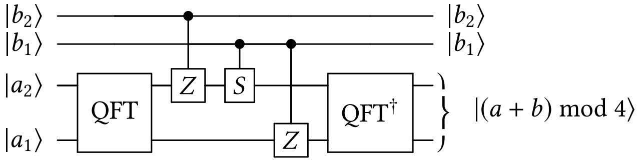


Fig. 1. Quantum circuit for performing Draper Adder on 2-bit integers (i.e, $n = 2, d = 4$)

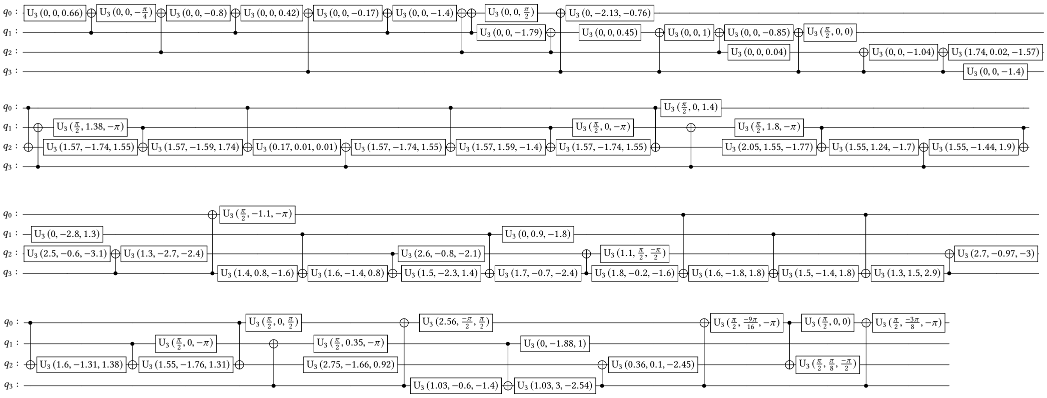


Fig. 2. Quantum circuit for performing Draper adder generated by qiskit-transpiler.

nonintuitive, making their semantic effects on data values difficult to comprehend. In light of this motivation, this paper investigates a method for automatically programming quantum circuits.

Problem: Component-based Quantum Circuit Synthesis. Specifically, we study the problem of synthesizing quantum circuits using user-provided components. As input, our quantum circuit synthesizer takes input-output state vectors and component quantum gates; as output, it generates a quantum circuit that implements the specified functionality in terms of the supplied gates. Note that our component-based synthesis differs from previous work on quantum circuit synthesis (also known as unitary synthesis). While the goal of earlier work [Davis et al. 2020; Goubault de Brugière et al. 2020; Shende et al. 2006; Tucci 2005; Younis et al. 2020] is to "compile" unitary matrices into circuits with a fixed set of low-level gates (e.g., $U_3(\theta, \phi, \psi)$), our objective is to generate a human-readable, high-level implementation that reveals the algorithm's inner workings.

Consider, for instance, the modular addition specified by the following input-output specification:

$$|b\rangle\,|a\rangle \mapsto |b\rangle\,|(a+b) \bmod d\rangle \text{ where } a, b \text{ are } n\text{-bit binary representations of integers} \tag{1}$$

where d and n are fixed integers. For example, when $d = 4$ and $n = 2$, the specification includes 16 state vector mappings such as $|11\rangle\,|01\rangle \mapsto |11\rangle\,|00\rangle$. As in classical circuit design, this adder plays a crucial part in quantum algorithms (e.g, Shor's factorization [Shor 1997] algorithm). Using classical carry gates and several ancilla bits, we may implement the adder in a reversible circuit. Suppose, however, that we are interested in implementing the adder in a more quantum-native manner by utilizing Quantum Fourier Transform (QFT) and phase operations (e.g., S, Z), so that addition is performed in place without ancilla bits. Figure 1 depicts such a circuit found by our synthesizer, which is indeed equivalent to the algorithm previously known as Draper Adder [Draper 2000].

By contrast, Figure 2 shows the circuit generated by qiskit-transpiler [Aleksandrowicz et al. 2019; Transpiler 2022]. As input, qiskit-transpiler takes the unitary matrix specified by (1), and produces the circuit in Figure 2 by applying Quantum Shannon Decomposition [Shende et al. 2006]. The resulting circuit does not help programmers grasp the idea of in-place addition; 107 low-level gates are used, and it is unclear how these gates are combined to perform the addition. We note that our approach and existing compilation approach have different purposes; the goal of compilation is to produce circuits that are directly executable on quantum computers while our goal is to generate circuits that can be read and maintained by humans.

Approach: Modular Synthesis of Quantum Circuits. In this paper, we present a novel, modular algorithm for synthesizing quantum circuits. We use enumerative search as our primary method for component-based synthesis. To speed up the method, however, we perform module-level search and pruning. Instead of naively enumerating candidate circuits by sequentially applying component gates, we apply certain groups of gate operations, called modules, at once. Furthermore, we direct this modular search in a certain way that bridges the gap between the current circuit and the given input-output specification. To this purpose, we define four distinct module attributes—entanglement, superposition, phasing, and boolean—by characterizing the attribute difference of state vectors. In Section 4, we formalize our module-level search algorithm and present a pruning method that is sound under practical assumptions.

Evaluation results show that our modular algorithm is highly effective at solving component-based quantum-circuit synthesis problems. For evaluation, we gathered 17 benchmark problems from various sources, including online forums and textbooks, and compared the performance of our module-level algorithm to that of a baseline algorithm that performs a conventional, gate-level enumerative search. Our algorithm successfully synthesized 16 out of 17 circuits in 96.6 seconds on average. The baseline algorithm, on the other hand, succeeded in 10 benchmarks, where our algorithm increased the speed of the baseline algorithm by 20.3x for those 10 problems.

Contributions. Our contributions are summarized as follows:

- We present a new modular approach to the component-based synthesis of quantum circuits. To our knowledge, our work provides the first method for synthesizing quantum circuits from arbitrary, user-supplied component gates.
- We experimentally show that our module-level synthesis algorithm is significantly more effective than a gate-level algorithm on a variety of benchmark problems. Our tool and benchmarks are publicly available:

https://github.com/kupl/qsyn

2 PRELIMINARIES

This section gives background information on quantum circuits and computation.

Qubits and Quantum States. Unlike classical bits, the state of a qubit can be in a superposition of two basis states. Using the Dirac notation, a single qubit is represented by a two-dimensional state vector of the form:

$$|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$$

where $|0\rangle$ and $|1\rangle$ denote the computational basis states:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \qquad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

and $\alpha_0, \alpha_1 \in \mathbb{C}$ are complex numbers, called *probability amplitudes*, such that $|\alpha_0|^2 + |\alpha_1|^2 = 1$. For example, $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$, $\frac{1}{\sqrt{2}}|0\rangle - \frac{i}{\sqrt{2}}|1\rangle$, $\frac{1}{2}|0\rangle + \frac{\sqrt{3}}{2}|1\rangle$ are legitimate states of a qubit.

The state of a two-qubit system is represented by a linear combination (superposition) of four basis states:

$$|\psi\rangle = \alpha_0 |00\rangle + \alpha_1 |01\rangle + \alpha_2 |10\rangle + \alpha_3 |11\rangle$$

where $\alpha_0, \alpha_1, \alpha_2, \alpha_3 \in \mathbb{C}$ are complex numbers such that $|\alpha_0|^2 + |\alpha_1|^2 + |\alpha_2|^2 + |\alpha_3|^2 = 1$ and the basis states, i.e., $|00\rangle$, $|01\rangle$, $|10\rangle$, and $|11\rangle$, represent the following state vectors:

$$|00\rangle = \begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \qquad |01\rangle = \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \qquad |10\rangle = \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \qquad |11\rangle = \begin{bmatrix}0\\0\\0\\1\end{bmatrix}.$$

In general, the state of an N-qubit system is defined with 2^N basis states, denoted $|x\rangle$ for $x \in \{0,1\}^N$, and is represented by a linear combination

$$|\psi\rangle = \sum_{x\in\{0,1\}^N} \alpha_x |x\rangle$$

where amplitudes, $\alpha_x \in \mathbb{C}$, are complex numbers such that $\sum_{x\in\{0,1\}^N} |\alpha_x|^2 = 1$. Each basis state $|x\rangle$ represents a 2^N-dimensional one-hot vector whose x-th element is 1.

The basis vectors for a multi-qubit system can be constructed by the tensor product of basis vectors for smaller quantum systems. For example, a basis vector $|01\rangle$ for a two-qubit system $\mathbf{q} = \{q_0, q_1\}$ is the tensor product of vector $|0\rangle$ and $|1\rangle$, i.e., $|01\rangle = |0\rangle \otimes |1\rangle$, as follows:

$$|01\rangle_{\mathbf{q}} = |0\rangle_{q_0} \otimes |1\rangle_{q_1} = \begin{bmatrix}1\\0\end{bmatrix} \otimes \begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$$

where we use the notation $|\psi\rangle_{\mathbf{q}}$ to indicate that the vector $|\psi\rangle$ denotes the state of qubits $\mathbf{q}$.

Properties of Quantum States. For an N-qubit system $|\psi\rangle = \sum_{x\in\{0,1\}^N} \alpha_x |x\rangle$, measuring the qubits returns one of classical states $|x\rangle$ with probability $|\alpha_x|^2$. Upon measurement, the state of the system collapses to the observed state $|x\rangle$. For example, by measuring the qubits in state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, we observe either $|00\rangle$ or $|11\rangle$ with equal probability, and the state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ collapes to the observed state.

Consider two quantum states $|\psi\rangle = \sum_x \alpha_x |x\rangle$ and $|\phi\rangle = \sum_y \beta_y |y\rangle$. Consider two amplitudes α_j and β_j of $|\psi\rangle$ and $|\phi\rangle$, respectively, for a certain index j. We say α_j and β_j differ by a *relative phase* if $\alpha_j = e^{i\theta}\beta_j$ for some $\theta \in (0, 2\pi)$. For example, consider $|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ and $|\phi\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, where the two amplitudes of $|1\rangle$ are related: $-\frac{1}{\sqrt{2}} = e^{i\pi}\frac{1}{\sqrt{2}}$. Hence, $|\psi\rangle$ and $|\phi\rangle$ said to be differerent in a relative phase. Note that quantum states that differ only in relative phases induce the same probability distribution of measurements since when $\alpha_j = e^{i\theta}\beta_j$ we have $|\alpha_j|^2 = |e^{i\theta}|^2|\beta_j|^2 = |\beta_j|^2$.

For quantum state $|\psi\rangle$ on qubit register $\mathbf{q}$, we say it is *entangled* in two sub-systems $\mathbf{q}_1, \mathbf{q}_2$ if it cannot be decomposed into tensor products of smaller two state vectors, i.e, for any $|\psi_1\rangle_{\mathbf{q}_1}, |\psi_2\rangle_{\mathbf{q}_2}$ such that $\mathbf{q}_1 \uplus \mathbf{q}_2 = \mathbf{q}$,

$$|\psi\rangle \neq |\psi_1\rangle_{\mathbf{q}_1} \otimes |\psi_2\rangle_{\mathbf{q}_2}.$$

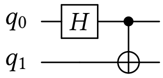
For example, the Bell state $|\Phi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ is entangled because we cannot find single-qubit states $|\psi_1\rangle, |\psi_2\rangle$ such that $|\Phi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle$.

Quantum Gates and Circuits. Let $\mathbf{q} = (q_0, q_1, \ldots, q_{N-1})$ be the qubits of an N-qubit system. A quantum circuit is a sequence of quantum gates:

$$C = G_1(\mathbf{q}_1); G_2(\mathbf{q}_2); \ldots; G_n(\mathbf{q}_n)$$

where each quantum gate $G_i(\mathbf{q}_i)$ consists of gate operation G_i and qubit register $\mathbf{q}_i \subseteq \mathbf{q}$ to which the operation G_i is applied. The gate operation G_i denotes a $2^{|\mathbf{q}_i|} \times 2^{|\mathbf{q}_i|}$ unitary matrix.

Quantum circuits are typically represented by circuit diagrams, where wires represent physical positions of qubits and blocks or symbols on wires denote quantum gates applied to the corresponding qubit(s). For example, the quantum circuit $C = H(q_0); CNOT(q_0, q_1)$ is represented by the following circuit diagram:



The circuit diagram should be read from left to right. The circuit consists of two quantum gate operations: the first operation is the Hadamard gate H on qubit q_0. The second gate operation is $CNOT$ (Controlled-NOT) where q_0 is the control qubit and q_1 is the target qubit. The H and $CNOT$ gates denote unitary matrices as follows:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \qquad CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

We can always extend $G_i(\mathbf{q}_i)$ to operation on $\mathbf{q}$, that is, a $2^{|\mathbf{q}|} \times 2^{|\mathbf{q}|}$ matrix. If the gate operation is applied to consecutive qubits $q_{n_1}, \ldots, q_{n_m}$ such that $n_i + 1 = n_{i+1}$ $(1 \leq i \leq m)$, we can extend it to be regarded as a $2^{|\mathbf{q}|} \times 2^{|\mathbf{q}|}$ matrix as follows:

$$\Big(\bigotimes_{q_i \text{ s.t } i<n_1} I(q_i)\Big) \otimes G(q_{n_1}, .., q_{n_m}) \otimes \Big(\bigotimes_{q_i \text{ s.t } i>n_m} I(q_i)\Big)$$

where I denotes the identity matrix. Otherwise, we can apply swap operations to make the gate operation apply to consecutive qubits temporarily, apply the identity-matrix tensor product as above, and again apply swap operations to restore the original qubit positions. In this manner, we generally assume that any gate operation is represented by a $2^{|\mathbf{q}|} \times 2^{|\mathbf{q}|}$ matrix. This assumption allows us to define the semantics of quantum circuit C using simple matrix multiplication without worrying about dimensions as follows:

$$C = G_n(\mathbf{q}_n) \times \cdots \times G_1(\mathbf{q}_1).$$

Note that in the matrix multiplication above, we read the gate operation sequence from right to left.

Other Notations. For an N-qubit state vector $|\psi\rangle = \sum_{x \in \{0,1\}^N} \alpha_x |x\rangle$, we write $\mathsf{amp}_{|\psi\rangle}$ and $\mathsf{dist}_{|\psi\rangle}$ for the tuple of amplitudes and the distribution of measurements, respectively:

$$\mathsf{amp}_{|\psi\rangle} = (\alpha_0, \ldots, \alpha_{2^N-1}), \qquad \mathsf{dist}_{|\psi\rangle} = (|\alpha_0|^2, \ldots, |\alpha_{2^N-1}|^2).$$

We also define equality between tuples, $=_{perm}$ and $\neq_{perm}$, as equality up to permutation. For n-tuples t_1 and t_2 (either amp or dist)

$$t_1 =_{perm} t_2 \iff \text{there exists a permutation } \sigma \text{ such that } \sigma(t_2) = t_1.$$

3 PROBLEM DEFINITION

In this paper, we tackle the problem of synthesizing quantum circuits using user-supplied component gates. We assume the following items are given:

- N: the size of the circuit to be synthesized (i.e., the number of qubits),
- $E = \{(|in_i\rangle, |out_i\rangle) \mid i = 1, \dots, k\}$: input-output examples, and
- $\mathcal{G}$: a set of user-provided gates.

Given $(N, E, \mathcal{G})$ as input, our goal is to automatically generate a quantum circuit C on N-qubit register $\mathbf{q} = \{q_0, q_1, \dots, q_{N-1}\}$ that satisfies E, only using the gates in $\mathcal{G}$. Formally, for finite index $I = \{1, .., m\}$ we aim to find a sequence of gate operations $C = G_1(\mathbf{q}_1); G_2(\mathbf{q}_2); \dots; G_m(\mathbf{q}_m)$ such that $G_i \in \mathcal{G}$, $\mathbf{q}_i \subseteq \mathbf{q}$, and satisfies E exactly as

$$|out_i\rangle = C\,|in_i\rangle \text{ for all } (|in_i\rangle, |out_i\rangle) \in E. \tag{2}$$

Note that we can sometimes relax the correctness criterion (2) to ignore global phase, that is, for some $\theta \in [0, 2\pi)$:

$$e^{i\theta}\,|out_i\rangle = C\,|in_i\rangle \text{ for all } (|in_i\rangle, |out_i\rangle) \in E. \tag{3}$$

For instance, this is a natural choice for certain classes of synthesis problems, namely *state preparation*, where the task is to build a circuit that prepares a certain quantum state $|\psi\rangle$ from classical state $|x\rangle$ (e.g., $|0\dots0\rangle$). In this case, the specification is given as the singleton set $E = \{(|x\rangle, |\psi\rangle)\}$. Also note that, when the specification is given as a unitary matrix U, the input-output examples are defined as $E = \{(|x\rangle, U|x\rangle) \mid x \in \{0,1\}^N\}$.

Example 3.1 (Running Example). Consider the following question posted on StackExchange [SE 2020c]:

"How to convert 3-qubit quantum state $|100\rangle$ into $|GHZ\rangle = \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$
using only Hadamard and $CNOT$ gates?"

This is a state preparation problem and can be translated into our problem definition as follows:

- $N = 3$
- $E = \{(|100\rangle, \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle))\}$
- $\mathcal{G} = \{H, CNOT\}$

Given these inputs, our goal is to synthesize a circuit such as one presented in Figure 3.

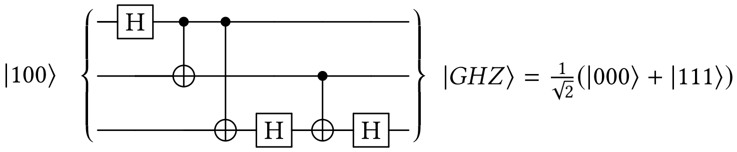
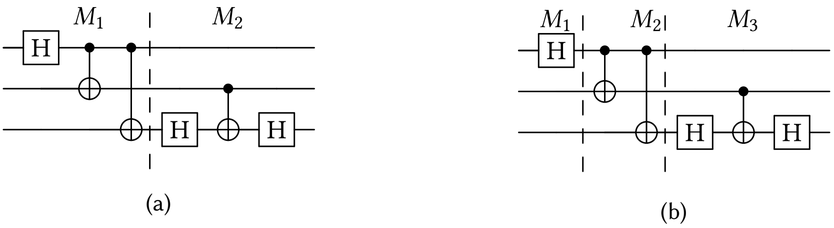

$|100\rangle$
H
H
H
$|GHZ\rangle = \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$

Fig. 3. Quantum circuit for transforming $|100\rangle$ to $|GHZ\rangle$

4 OUR SYNTHESIS ALGORITHM

In this section, we present a modular algorithm for synthesizing quantum circuits. Section 4.1 defines what we mean by modules and describe their properties. Section 4.2 describes the high-level structure of our algorithm. Then, we provide the details of the two crucial parts of our algorithm: module-level pruning (Section 4.3) and candidate module generation (Section 4.4). Finally, we discuss on complexity of our synthesis algorithm in Section 4.5.

4.1 Modular Representation of Quantum Circuits



(a)

(b)

Fig. 4. Modular representation of construction `GHZ_from_100`

Modules. Given a quantum circuit $C = G_1(\mathbf{q}_1); G_2(\mathbf{q}_2); \ldots; G_n(\mathbf{q}_n)$, we define a *module* of C to be a consecutive slice $G_l(\mathbf{q}_l); G_{l+1}(\mathbf{q}_{l+1}); \cdots; G_{l+m}(\mathbf{q}_{l+m})$ $(1 \le l \le n, 0 \le m \le n-l)$. For example, we can decompose the circuit in Figure 3 into two modules $C = M_1; M_2$ in Figure 4a, where

$$M_1 = H(q_0); CNOT(q_0, q_1); CNOT(q_0, q_2), \quad M_2 = H(q_2); CNOT(q_1, q_2); H(q_2).$$

A circuit may have multiple modular representations. For example, we can also decompose the circuit in Figure 3 into three modules $C = M_1; M_2; M_3$ in Figure 4b, where

$$M_1 = H(q_0), \quad M_2 = CNOT(q_0, q_1); CNOT(q_0, q_2), \quad M_3 = H(q_2); CNOT(q_1, q_2); H(q_2).$$

Attribute Difference. For two state vectors $|\psi\rangle$ and $|\phi\rangle$, we characterize their attribute difference, denoted $|\psi\rangle \ominus |\phi\rangle$. We consider four kinds of attributes:

$$\Omega = \{\textsc{Entangle}, \textsc{Superposition}, \textsc{Phasing}, \textsc{Bool}\}.$$

Intuitively, $|\psi\rangle \ominus |\phi\rangle$ is defined to be

- Entangle if $|\psi\rangle$ and $|\phi\rangle$ are different in how qubits are entangled,
- Superposition if $|\psi\rangle$ and $|\phi\rangle$ are different in how their basis states are superposed,
- Phasing if $|\psi\rangle$ and $|\phi\rangle$ differ in relative phases, and
- Bool if $|\psi\rangle$ and $|\phi\rangle$ differ classically.

To formally define $|\psi\rangle \ominus |\phi\rangle$, let us first define the notion of the entanglement partition of a state vector.

Definition 4.1 (Entanglement Partition). Let $|\psi\rangle_{\mathbf{q}}$ be an N-qubit state vector on qubit register $\mathbf{q}$. An entanglement partition of $|\psi\rangle_{\mathbf{q}}$ is the finest partition $\{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_k\}$ of the qubit register $\mathbf{q}$ such that $|\psi\rangle$ is separable as

$$|\psi\rangle_{\mathbf{q}} = |\psi_1\rangle_{\mathbf{q}_1} \otimes \cdots \otimes |\psi_k\rangle_{\mathbf{q}_k}$$

where each state vector $|\psi_i\rangle$ on $\mathbf{q}_i$ is fully entangled: for any $\mathbf{q}_i^{(1)}, \mathbf{q}_i^{(2)} \subseteq \mathbf{q}_i$ such that $\mathbf{q}_i^{(1)} \uplus \mathbf{q}_i^{(2)} = \mathbf{q}_i$ and state vectors $|\phi_1\rangle_{\mathbf{q}_i^{(1)}}$ and $|\phi_2\rangle_{\mathbf{q}_i^{(2)}}$, $|\psi_i\rangle$ is not decomposed into $|\phi_1\rangle$ and $|\phi_2\rangle$, i.e., $|\psi_i\rangle \neq |\phi_1\rangle \otimes |\phi_2\rangle$.

Example 4.2. The Bell state $|\Phi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ on qubit register $\mathbf{q} = \{q_0, q_1\}$ is entangled with entanglement partition $\{\{q_0, q_1\}\}$. Similarly, the entanglement partition of $|GHZ\rangle = \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$ is $\{\{q_0, q_1, q_2\}\}$. On the other hand, classic state vectors such as $|\phi\rangle = |100\rangle$ on qubit register $\mathbf{q} = \{q_0, q_1, q_2\}$ is not entangled at all; its entanglement partition is $\{\{q_0\}, \{q_1\}, \{q_2\}\}$.

Now we can define the attribute difference, $|\psi\rangle \ominus |\phi\rangle$, between state vectors.

Definition 4.3 (Attribute Difference between State Vectors). Consider two (different) state vectors $|\psi\rangle$ and $|\phi\rangle$ on a qubit register $\mathbf{q}$. The attribute difference between $|\psi\rangle$ and $|\phi\rangle$, denoted $|\psi\rangle \ominus |\phi\rangle$, is defined as follows:

- $|\psi\rangle \ominus |\phi\rangle = \textsc{Entangle}$ if the entanglement partitions of $|\psi\rangle$ and $|\phi\rangle$ are different,
- $|\psi\rangle \ominus |\phi\rangle = \textsc{Superposition}$ (or in short Sp) if $\mathsf{dist}_{|\psi\rangle} \neq_{perm} \mathsf{dist}_{|\phi\rangle}$,
- $|\psi\rangle \ominus |\phi\rangle = \textsc{Phasing}$ if $\mathsf{dist}_{|\psi\rangle} =_{perm} \mathsf{dist}_{|\phi\rangle}$ but $\mathsf{amp}_{|\psi\rangle} \neq_{perm} \mathsf{amp}_{|\phi\rangle}$, and
- $|\psi\rangle \ominus |\phi\rangle = \textsc{Bool}$ if $\mathsf{amp}_{|\psi\rangle} =_{perm} \mathsf{amp}_{|\phi\rangle}$ (by non-trivial permutation).

Although it is not a 'difference', we additionally define to be $|\psi\rangle \ominus |\phi\rangle = \textsc{Identity}$ (in short Id) if $|\psi\rangle = |\phi\rangle$.

There are two remarks regarding the definition:

- Superposition, Phasing, and Bool are mutually exclusive: when $|\psi\rangle \ominus |\phi\rangle \neq \textsc{Entangle}$, $|\psi\rangle \ominus |\phi\rangle$ is only one of Superposition, Phasing, and Bool. In this case, we say the attribute difference is local. However, when the difference is Entangle, $|\psi\rangle \ominus |\phi\rangle$ could still be one of $\{\textsc{Superposition}, \textsc{Phasing}, \textsc{Bool}\}$. For example, $|\phi\rangle = \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle)$ and $|\psi\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |11\rangle)$ differ in Entangle and Phasing. In this case, we prioritize Entangle and define $|\psi\rangle \ominus |\phi\rangle = \textsc{Entangle}$.
- When defining Phasing, unlike the usual definition of a relative phase, we compare distributions and amplitudes up to permutation. This allows us to view phasing operations in a broader sense. For example, the $Z \times X$ operation is still regarded as a phasing operation according to our definition.

Example 4.4. Consider $|GHZ\rangle = \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$. We show several examples of attribute difference, $|GHZ\rangle \ominus |\phi\rangle$, between $|GHZ\rangle$ and a state vector $|\phi\rangle$.

(1) When $|\phi\rangle = |100\rangle$, $|GHZ\rangle \ominus |\phi\rangle = \textsc{Entangle}$. State vector $|100\rangle$ is not entangled at all, so its entanglement partition is $\{\{q_0\}, \{q_1\}, \{q_2\}\}$. On the other hand, $|GHZ\rangle$ is fully entangled and the entanglement partition is $\{\{q_0, q_1, q_2\}\}$. Thus, their entanglement paritions are different.

(2) When $|\phi\rangle = \frac{1}{\sqrt{3}}(|001\rangle + |010\rangle + |100\rangle)$, $|GHZ\rangle \ominus |\phi\rangle = \textsc{Superposition}$. Note that the entanglement partition of $|\phi\rangle$ is the same as $|GHZ\rangle$. Thus, we compare the distributions of the state vectors:

$$\mathsf{dist}_{|GHZ\rangle} = (\frac{1}{2}, 0, 0, 0, 0, 0, 0, \frac{1}{2}), \qquad \mathsf{dist}_{|\phi\rangle} = (0, \frac{1}{3}, \frac{1}{3}, 0, \frac{1}{3}, 0, 0, 0).$$

Apparently, $\mathsf{dist}_{|GHZ\rangle} \neq_{perm} \mathsf{dist}_{|\phi\rangle}$.

(3) When $|\phi\rangle = \frac{1}{\sqrt{2}}(|000\rangle - |111\rangle)$, $|GHZ\rangle \ominus |\phi\rangle = \textsc{Phasing}$ since the two states have the same distribution, $(\frac{1}{2}, 0, 0, 0, 0, 0, 0, \frac{1}{2})$, but their amplitudes are different:

$$\mathsf{amp}_{|GHZ\rangle} = (\frac{1}{\sqrt{2}}, 0, 0, 0, 0, 0, 0, \frac{1}{\sqrt{2}}), \qquad \mathsf{amp}_{|\phi\rangle} = (\frac{1}{\sqrt{2}}, 0, 0, 0, 0, 0, 0, -\frac{1}{\sqrt{2}}).$$

(4) When $|\phi\rangle = \frac{1}{\sqrt{2}}(|001\rangle + |110\rangle)$, $|GHZ\rangle \ominus |\phi\rangle = \textsc{Bool}$ because the amplitudes of $|GHZ\rangle$ and $|\phi\rangle$ are equivalent up to permutation:

$$\mathsf{amp}_{|GHZ\rangle} = (\frac{1}{\sqrt{2}}, 0, 0, 0, 0, 0, 0, \frac{1}{\sqrt{2}}), \qquad \mathsf{amp}_{|\phi\rangle} = (0, \frac{1}{\sqrt{2}}, 0, 0, 0, 0, \frac{1}{\sqrt{2}}, 0).$$

Indeed, we can reach from $|\phi\rangle$ to $|GHZ\rangle$ by $X(q_2)$, that is, $X(q_2)\,|\phi\rangle = |GHZ\rangle$.

Attribute of Modules. From the definition of attribute difference between state-vectors, attributes can also be defined for quantum operators and modules acting on state-vectors.

Definition 4.5 (Attribute of Module or Quantum Operator). Given a quantum operator U and an input state-vector $|in\rangle$, we define the attribute of U w.r.t. $|in\rangle$, denoted $Att_{|in\rangle}(U)$, to be $U\,|in\rangle \ominus |in\rangle$. For a modular quantum circuit $C = M_1; \cdots ; M_k$ and input $|in\rangle$, the attribute of module M_l, denoted $Att_{M_{l-1}\ldots M_1|in\rangle}(M_l)$, is defined to be $(M_l \cdots M_1)\,|in\rangle \ominus (M_{l-1} \cdots M_1)\,|in\rangle$.

Example 4.6. Consider the modularized circuit in Figure 4a with input state-vector $|in\rangle = |100\rangle$.

- $Att_{|100\rangle}(M_1) = M_1\,|100\rangle \ominus |100\rangle = \frac{1}{\sqrt{2}}(|000\rangle - |111\rangle) \ominus |100\rangle =$ Entangle, where module $M_1 = H(q_0); CNOT(q_0, q_1); CNOT(q_0, q_2)$ maps $|100\rangle$ to $\frac{1}{\sqrt{2}}(|000\rangle - |111\rangle)$. Intuitively, the attribute of M_1 is Entangle because its input $|100\rangle$ and output $\frac{1}{\sqrt{2}}(|000\rangle - |111\rangle)$ have different engtanglement partitions.
- $Att_{M_1|100\rangle}(M_2) = (M_2 \times M_1)\,|100\rangle \ominus M_1\,|100\rangle = |GHZ\rangle \ominus \frac{1}{\sqrt{2}}(|000\rangle - |111\rangle) =$ Phasing, where module $M_2 = H(q_2); CNOT(q_1, q_2); H(q_2)$ maps $\frac{1}{\sqrt{2}}(|000\rangle - |111\rangle)$ to $|GHZ\rangle = \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$. These two state-vectors differ only in a relative phase.
- $Att_{|100\rangle}(M_2 \times M_1) = (M_2 \times M_1)\,|100\rangle \ominus |100\rangle = |GHZ\rangle \ominus |100\rangle =$ Entangle.

We define an order between modules based on their attributes as follows.

Definition 4.7 (Order Between Modules). Consider a modularized quantum circuit $C = M_1; \cdots ; M_k$. Given two consecutive modules M_i, M_{i+1} $(1 \le i < k)$ and input $|in\rangle$ to M_i, where the input vector will be clear from the context:

$$M_i \succ M_{i+1} \iff Att_{|in\rangle}(M_{i+1} \times M_i) = Att_{|in\rangle}(M_i)$$

On the other hand, we define $M_i \prec M_{i+1}$ if M_{i+1} absorbs the attribute of M_i:

$$M_i \prec M_{i+1} \iff Att_{|in\rangle}(M_{i+1} \times M_i) = Att_{M_i|in\rangle}(M_{i+1})$$

We write $M_i \simeq M_{i+1}$ when both $M_i \succ M_{i+1}$ and $M_i \prec M_{i+1}$ hold.

Example 4.8. In Example 4.6, $M_1 \succ M_2$ holds because $Att_{|100\rangle}(M_2 \times M_1) =$ Entangle $= Att_{|100\rangle}(M_1)$. However, $M_1 \nprec M_2$ since $Att_{|100\rangle}(M_2 \times M_1) =$ Entangle $\neq$ Phasing $= Att_{M_1|100\rangle}(M_2)$. Hence, $M_1 \not\simeq M_2$ and $M_1 \gtrsim M_2$.

Example 4.9. Consider a quantum circuit $C = M_1; M_2$, where $M_1 = X(q_0); H(q_0)$; and $M_2 = H(q_0)$. For input vector $|0\rangle$, $Att_{|0\rangle}(M_1) =$ Superposition and $Att_{M_1|0\rangle}(M_2) =$ Superposition. However, $Att_{|0\rangle}(M_2 \times M_1) = Att_{|0\rangle}(X(q_0)) =$ Bool, which is neither $Att_{|0\rangle}(M_1)$ nor $Att_{M_1|0\rangle}(M_2)$. Hence, $M_1 \nsucc M_2$ and $M_1 \nprec M_2$, which implies that $\succ$ is a partial order.

Definition 4.10. We define the order $<$ between attributes as follows:

$$\text{Identity} < \text{Bool} < \text{Phasing} < \text{Superposition} < \text{Entangle} \tag{4}$$

The order $\succ$ between modules and the order $<$ between attributes are closely related.

Proposition 4.11. *For $\omega, \omega' \in \Omega = \{$Entangle, Superposition, Phasing, Bool, Identity$\}$, consider modules M_i, M_{i+1} with input $|in\rangle$ (for M_i) such that $Att_{|in\rangle}(M_i) = \omega$ and $Att_{M_i|in\rangle}(M_{i+1}) = \omega'$. Then,*

$$\omega > \omega' \iff M_i \gtrsim M_{i+1}.$$

Proof. By case analysis on $\omega \in \Omega$. □

We say a modularized quantum circuit $C = M_1; \cdots ; M_k$ is *monotonically decreasing* (resp., *strictly decreasing*) if $M_i > M_{i+1}$ (resp., $M_i \gtrsim M_{i+1}$) holds for every pair M_i, M_{i+1} $(1 \leq i \leq k-1)$ of consecutive modules.

As we will show later, these properties characterize the quantum circuit we are constructing, concretely for the state preparation case.

Example 4.12. The modularized circuit in Figure 4a is strictly decreasing because $M_1 \gtrsim M_2$ (see Example 4.9). We can also check this by Proposition 4.11 : since $Att_{|100\rangle}(M_1) = \textsc{Entangle} > \textsc{Phasing} = Att_{M_1|100\rangle}(M_2)$, it is strictly decreasing. In contrast, the modular representation in Figure 4b is not decreasing since the attributes of M_1, M_2, and M_3 are Superposition, Entangle, and Phasing, respectively, but Superposition < Entangle.

4.2 Algorithm Outline

Algorithm 1 describes the outline of our modular synthesis algorithm. Let $(N, E, \mathcal{G})$ be the problem specification, where N is the circuit size, E input-output examples, and $\mathcal{G}$ the user-provided gates.

High-Level Structure. The main feature of our algorithm is that it works at a module-level, rather than a gate-level. That is, the algorithm uses a set $\mathcal{M}$ of modules, not gates, as (atomic) components, and aims to find a sequence $C = M_1; M_2; \ldots; M_l$ of modules that satisfies input-output examples. Thus, the very first job of the algorithm is to build the set $\mathcal{M}$ of component modules from the gate set $\mathcal{G}$. For the moment, assume $\mathcal{M}_k$ is given (where k is some parameter that determines space of $\mathcal{M}$, explained in Section 4.4) and let $\mathcal{M}_{C,E} \subseteq \mathcal{M}_k$ be the subset of modules relevant for the current context (the circuit C and examples E). We explain how to build $\mathcal{M}_k$ and $\mathcal{M}_{C,E}$ in Section 4.4.

We explain Algorithm 1 line by line. The algorithm performs enumerative search with pruning at the module-level. It is a worklist-based algorithm, where the worklist W is a set of (partial) circuits and initially contains the empty circuit ϵ (line 1). The synthesis loop at lines 2–11 is repeated until W becomes empty or the time budget expires. At lines 3 and 4, a circuit C is chosen from the worklist. For Choose, we prioritize $C \in W$ with smallest modular length and number of gate operations, in order to expect a smallest size circuit as synthesis result. At lines 5–6, we consider each candidate module $M \in \mathcal{M}_{C,E}$ and use the predicate

$$\mathsf{is\_gap\_filled}(C, M, E) \in \{True, False\}$$

to see if module M "fills" the attribute gap between the current circuit C and the desired one specified by examples E. (The definition of is_gap_filled will be given shortly in Section 4.3.) When is_gap_filled(C, E, M) evaluates to $True$, we append M to C (line 7) and check whether the extended circuit C' is a solution satisfying all input-output examples (possibly up to global phase by user's choice) (line 8):

$$\mathsf{solution}(C, E) \iff \forall(|in\rangle, |out\rangle) \in E : C\,|in\rangle = |out\rangle\,.$$

The algorithm ends if a solution is discovered (line 9). Otherwise, it repeats the previous steps while adding C' to the worklist W (line 10).

Example. Let us illustrate the algorithm for the problem in Example 3.1, where the goal is to synthesize the GHZ circuit in Figure 3 given $E = \{(|100\rangle, \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle))\}$ and $\mathcal{G} = \{H, CNOT\}$. The first step of our algorithm is to generate the set of modules from $\mathcal{G}$. For simplictiy, suppose the following three modules are available for use regardless of given circuit C:

$$\mathcal{M} = \left\{ \begin{array}{lcl} M_1 & = & H(q_0); CNOT(q_0, q_1); CNOT(q_0, q_2), \\ M_2 & = & CNOT(q_0, q_1), \\ M_3 & = & H(q_2); CNOT(q_1, q_2); H(q_2) \end{array} \right\}$$

Algorithm 1 Algorithm Outline

```
Input: $N$: circuit size, $E$: input-output examples, $\mathcal{G}$: user-defined gate set
Output: Quantum circuit $C$ satisfying $E$
 1: $W \leftarrow \{\epsilon\}$
 2: repeat
 3:     $C \leftarrow \mathsf{Choose}(W)$
 4:     $W \leftarrow W \setminus \{C\}$
 5:     for $M \in \mathcal{M}_{C,E}$ do        ▷ $\mathcal{M}_{C,E}$: component modules relevant for $C$ and $E$ (Section 4.4)
 6:         if is_gapfilled$(C, M, E)$ then        ▷ Attribute-based pruning (Section 4.3)
 7:             $C' \leftarrow C; M$
 8:             if solution$(C', E)$ then
 9:                 return $C'$
10:             $W \leftarrow W \cup \{C'\}$
11: until $W = \emptyset$ or timeout
```

Then, our algorithm goes into the main loop and works as follows:

(1) The algorithm begins with the blank circuit, $C = \epsilon$, which acts as the identify function on the input state $|100\rangle$, i.e., $C\,|100\rangle = |100\rangle$. At line 5, we consider each of $\mathcal{M} = \{M_1, M_2, M_3\}$ and use is_gap_filled to identify modules that fill the gap between C and the desired solution. To do so, we first find out that the attribute difference between C and the solution is Entangle because $\frac{1}{\sqrt{2}}(|000\rangle + |111\rangle) \ominus C\,|100\rangle$ = Entangle according to Definition 4.3. Next, we compute the module attributes (Definition 4.5) for M_1, M_2, and M_3 as follows:

$$Att_{C|100\rangle}(M_1) = (M_1 \times C)\,|100\rangle \ominus C\,|100\rangle = \tfrac{1}{\sqrt{2}}(|000\rangle - |111\rangle) \ominus |100\rangle = \textsc{Entangle}$$
$$Att_{C|100\rangle}(M_2) = (M_2 \times C)\,|100\rangle \ominus C\,|100\rangle = |110\rangle \ominus |100\rangle = \textsc{Bool}$$
$$Att_{C|100\rangle}(M_3) = (M_3 \times C)\,|100\rangle \ominus C\,|100\rangle = |100\rangle \ominus |100\rangle = \textsc{Identity}$$

Note that the attribute of module M_1, i.e., $Att_{C|100\rangle}(M_1)$ = Entangle, is equivalent to the attribute difference, Entangle, between C and the solution. Thus, is_gap_filled(C, M_1, E) at line 6 evlauates to *True* and we append M_1 to C. (For M_2 and M_3, is_gap_filled evaluates to *False*.) At line 8, we check if $C' = C; M_1 = M_1$ can be a solution. In this case, since $M_1\,|000\rangle = \frac{1}{\sqrt{2}}(|000\rangle - |111\rangle) \neq \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$, solution$(C', E)$ evaluates to *False*.

(2) In the second iteration, the current circuit C is M_1 and we try to extend it with modules in $\mathcal{M}$. To do so, we again find out that the current attribute difference between C and the solution is Phasing:

$$\frac{1}{\sqrt{2}}(|000\rangle + |111\rangle) \ominus C\,|100\rangle = \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle) \ominus \frac{1}{\sqrt{2}}(|000\rangle - |111\rangle) = \textsc{Phasing}$$

and compare it with the attributes of modules M_1, M_2, and M_3:

$$\begin{aligned} Att_{M_1|100\rangle}(M_1) &= (M_1 \times M_1)\,|100\rangle \ominus M_1\,|100\rangle \\ &= \tfrac{1}{2}(|000\rangle - |011\rangle + |100\rangle + |111\rangle) \ominus \tfrac{1}{\sqrt{2}}(|000\rangle - |111\rangle) = \textsc{Superposition} \\ Att_{M_1|100\rangle}(M_2) &= (M_2 \times M_1)\,|100\rangle \ominus M_1\,|100\rangle \\ &= \tfrac{1}{\sqrt{2}}(|000\rangle - |101\rangle) \ominus \tfrac{1}{\sqrt{2}}(|000\rangle - |111\rangle) = \textsc{Entangle} \\ Att_{M_1|100\rangle}(M_3) &= (M_3 \times M_1)\,|100\rangle \ominus M_1\,|100\rangle \\ &= \tfrac{1}{\sqrt{2}}(|000\rangle + |111\rangle) \ominus \tfrac{1}{\sqrt{2}}(|000\rangle - |111\rangle) = \textsc{Phasing} \end{aligned}$$

In this case, because the attribute of M_3 coincides with the current difference, we append M_3 to $C = M_1$. The resulting circuit $C' = M_1; M_3$ satisfies the example specification:

$$(M_3 \times M_1) |100\rangle = \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$$

and the algorithm terminates with the solution found.

4.3 Pruning

Given an example $(|in\rangle, |out\rangle) \in E$ and a quantum circuit C, let $\text{gap\_att}_{|in\rangle,|out\rangle}(C)$ be the attribute difference between C and the circuit whose behavior is specified by the example $(|in\rangle, |out\rangle)$:

$$\text{gap\_att}_{|in\rangle,|out\rangle}(C) \overset{def}{=} |out\rangle \ominus C\,|in\rangle\,.$$

Single IO. When $E = \{(|in\rangle, |out\rangle)\}$ is a singleton set, we append a module M to the current circuit C if the attribute difference between C and E coincides with the attribute of M, i.e., $\text{gap\_att}_{|in\rangle,|out\rangle}(C) = Att_{C|in\rangle}(M)$. That is, we define is_gap_filled as follows:

$$\text{is\_gap\_filled}(C, M, (|in\rangle, |out\rangle)) \iff \text{gap\_att}_{|in\rangle,|out\rangle}(C) = Att_{C|in\rangle}(M). \tag{5}$$

Example 4.13. Suppose $E = \{|100\rangle \mapsto |GHZ\rangle\}$ and $C = \epsilon$. Consider modules M_1 and M_2 in Figure 4a. In this case, we can append M_1 to C because

$$\text{gap\_att}_{|100\rangle,|GHZ\rangle}(C) = \textsc{Entangle} = Att_{C|100\rangle}(M_1).$$

In contrast, we do not append M_2 to C because

$$\text{gap\_att}_{|100\rangle,|GHZ\rangle}(C) = \textsc{Entangle} \neq \textsc{Identity} = Att_{C|100\rangle}(M_2).$$

Multi IO. When $E = \{(|in_i\rangle, |out_i\rangle) \mid i = 1, \ldots, k\}$ is not a singleton set. We may want to append a module M to the current circuit C if the module fills the gap for every example:

$$\forall i \in [1, k] : \text{gap\_att}_{|in_i\rangle,|out_i\rangle}(C) = Att_{C|in_i\rangle}(M).$$

However, we found that modules that fill multiple gaps are often complex (i.e., requiring many gate operations) and are hardly captured by a module of small depths. Instead, we allow modules to be appended even if they partially fill the gaps:

$$\text{is\_gap\_filled}(C, M, E) \iff \left[\sum_{(|in\rangle,|out\rangle)\in E} \mathbf{1}_{\text{is\_gap\_filled}(C,M,(|in\rangle,|out\rangle))}\right] \geq \delta \tag{6}$$

where δ is a hyper-parameter and we set $\delta = \lfloor |E|/2 \rfloor$ in our implementation.

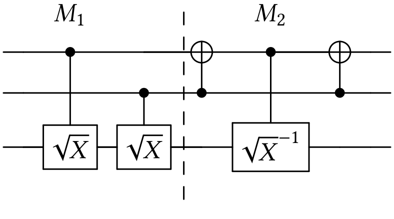


Fig. 5. Implementation of the Toffoli gate using $\mathcal{G} = \{C\sqrt{X}, C\sqrt{X}^{-1}, CNOT\}$

Example 4.14. Consider the problem of synthesizing the Toffoli gate (i.e., the $CCNOT$ gate) using $\mathcal{G} = \{C\sqrt{X}, C\sqrt{X}^{-1}, CNOT\}$ as component gates, where input-output examples are given as $E = \{|000\rangle \mapsto |000\rangle, |001\rangle \mapsto |001\rangle, |010\rangle \mapsto |010\rangle, |011\rangle \mapsto |011\rangle, |100\rangle \mapsto |100\rangle, |101\rangle \mapsto |101\rangle, |110\rangle \mapsto |111\rangle, |111\rangle \mapsto |110\rangle\}$.

The solution circuit is presented in Figure 5. The circuit C consists of two modules: $C = M_1; M_2$. The action of each module is illustrated as follows:

$$\begin{array}{ccccc}
 & M_1 & & M_2 & \\
|000\rangle & \mapsto & |000\rangle & \mapsto & |000\rangle \\
|001\rangle & \mapsto & |001\rangle & \mapsto & |001\rangle \\
|010\rangle & \mapsto & (\frac{1}{2} - \frac{1}{2}i)\,|010\rangle + (\frac{1}{2} + \frac{1}{2}i)\,|011\rangle & \mapsto & |010\rangle \\
|011\rangle & \mapsto & (\frac{1}{2} + \frac{1}{2}i)\,|010\rangle + (\frac{1}{2} - \frac{1}{2}i)\,|011\rangle & \mapsto & |011\rangle \\
|100\rangle & \mapsto & (\frac{1}{2} - \frac{1}{2}i)\,|100\rangle + (\frac{1}{2} + \frac{1}{2}i)\,|101\rangle & \mapsto & |100\rangle \\
|101\rangle & \mapsto & (\frac{1}{2} + \frac{1}{2}i)\,|100\rangle - (\frac{1}{2} + \frac{1}{2}i)\,|101\rangle & \mapsto & |101\rangle \\
|110\rangle & \mapsto & |111\rangle & \mapsto & |111\rangle \\
|111\rangle & \mapsto & |110\rangle & \mapsto & |110\rangle
\end{array}$$

Module M_1 satisfies the condition in (6), i.e., $\text{is\_gap\_filled}(\epsilon, M_1, E) = \mathit{True}$, because it fills the gap of half of input specs: when $(|in\rangle, |out\rangle) \in \{|000\rangle \mapsto |000\rangle, |001\rangle \mapsto |001\rangle, |110\rangle \mapsto |111\rangle, |111\rangle \mapsto |110\rangle\}$, $\text{is\_gap\_filled}(\epsilon, M_1, (|in\rangle, |out\rangle)) = \mathit{True}$ holds since

$$\text{gap\_att}_{|in\rangle,|out\rangle}(\epsilon) = \textsc{Bool} = Att_{\epsilon|in\rangle}(M_1).$$

However, when $(|in\rangle, |out\rangle) \in \{|010\rangle \mapsto |010\rangle, |011\rangle \mapsto |011\rangle, |100\rangle \mapsto |100\rangle, |101\rangle \mapsto |101\rangle\}$, $\text{is\_gap\_filled}(\epsilon, M_1, (|in\rangle, |out\rangle))$ evaluates to *False* because

$$\text{gap\_att}_{|in\rangle,|out\rangle}(\epsilon) = \textsc{Id} \neq \textsc{Superposition} = Att_{\epsilon|in\rangle}(M_1).$$

Note that, in the latter case, M_1 introduces a new attribute difference, Superposition, which is filled by the next module M_2, i.e., $\text{is\_gap\_filled}(M_1, M_2, (|in\rangle, |out\rangle))) = \mathit{True}$ for all $(|in\rangle, |out\rangle) \in E$. For example, $\text{is\_gap\_filled}(M_1, M_2, (|010\rangle, |010\rangle)) = \mathit{True}$ because

$$\text{gap\_att}_{|010\rangle,|010\rangle}(M_1) = \textsc{Superposition} = Att_{M_1|010\rangle}(M_2).$$

Optimization. The $\ominus$ operation (Definition 4.3) can be expensive in practice. Thus, we provide an optimized version of is_gap_filled in (5), which reduce calls to the $\ominus$ operation:

$$\text{is\_gap\_filled}^{opt}(C, M, (|in\rangle, |out\rangle)) \overset{def}{=}$$

$$\begin{cases} \text{gap\_att}_{|in\rangle,|out\rangle}(C) > \text{gap\_att}_{|in\rangle,|out\rangle}(C; M) & \text{if } \text{gap\_att}_{|in\rangle,|out\rangle}(C) \neq \textsc{Bool}, \textsc{Identity} \\ \text{gap\_att}_{|in\rangle,|out\rangle}(C) \geq \text{gap\_att}_{|in\rangle,|out\rangle}(C; M) & \text{otherwise} \end{cases} \tag{7}$$

Note that is_gap_filled^{opt} replaces the query $Att_{C|in\rangle}(M)$ in (5) by $\text{gap\_att}_{|in\rangle,|out\rangle}(C; M)$. Since we can cache the result of gap_att, is_gap_filled^{opt} does not require re-calculating gap_att for $(C; M)$ in the future when $\text{is\_gap\_filled}^{opt}(C, M, (|in\rangle, |out\rangle))$ evaluates to true and $C; M$ gets into the worklist. The following proposition shows that is_gap_filled^{opt} in (7) implies is_gap_filled in (5).

Proposition 4.15. *For any quantum circuit C, module M, and example $(|in\rangle, |out\rangle)$ (where $Att_{C|in\rangle}(M) \neq \textsc{Id}$),*

$$\text{is\_gap\_filled}^{opt}(C, M, (|in\rangle, |out\rangle)) \implies \text{is\_gap\_filled}(C, M, (|in\rangle, |out\rangle)).$$

Proof. By case analysis on $\text{gap\_att}_{|in\rangle,|out\rangle}(C)$.

□

A consequence of Proposition 4.15 is that is_gap_filled^{opt} more aggressively prunes out modules than is_gap_filled.

Soundness. Our pruning procedures, is_gap_filled and is_gap_filled^{opt}, are sound when the solution circuit to be synthesized satisfies some conditions. Below, let C^* be the solution circuit with modularization $C^* = M_1; \ldots ; M_k$. Also, let $|\psi_i\rangle$ be a state vector used as input to M_i, that is, $|\psi_i\rangle = M_{i-1} \cdots M_1 |in\rangle$ and $|\psi_1\rangle = |in\rangle$.

We first prove the soundness of is_gap_filled^{opt} since it is mainly used in practice. For the optimized criterion is_gap_filled^{opt} to be sound, we require that the solution circuit is strictly decreasing (with allowance of sequential Bool modules in tail):

$$Att_{|\psi_1\rangle}(M_1) > Att_{|\psi_2\rangle}(M_2) > \cdots > Att_{|\psi_k\rangle}(M_k).$$

The following theorem states the soundness of is_gap_filled^{opt}.

Theorem 4.16. *Let $E = \{(|in\rangle, |out\rangle)\}$ be an example and $C^* = M_1; \cdots ; M_k$ ($M_i \in \mathcal{M}$ and attribute of each M_i is not Identity) be the solution circuit to be synthesized such that $C^*(|in\rangle) = |out\rangle$. Suppose C^* is strictly decreasing (by input $|in\rangle$) with allowance of sequential Bool modules. Then, for any prefix $C = M_1; \cdots ; M_{l-1}$ ($l \leq k$) of C^*,*

$$\text{is\_gap\_filled}^{opt}(C, M_l, (|in\rangle, |out\rangle)) = True.$$

Proof. Note that for any consecutive subsequence of strict decreasing modules $M_p; ..; M_{p+n}$, it is guaranteed that the head module M_p absorbs the attribute of following modules $M_{p+1}, .., M_{p+n}$ as

$$Att_{M_{p-1} \cdot\cdot M_1 |in\rangle}(M_{p+n} \cdots M_p) = Att_{M_{p-1} \cdot\cdot M_1 |in\rangle}(M_p).$$

(this can be shown by considering cases on $Att_{M_{p-1} \cdot\cdot M_1 |in\rangle}(M_p)$). Let $|\psi\rangle = C |in\rangle$ and b ($1 \leq b \leq k$) be the index of the first module whose attribute is Bool. We consider three cases for b as follows. Suppose $l - 1 < b - 2$. This means that the attributes of M_l and M_{l+1} are not Bool. Then, module sequences $M_l; M_{l+1}; ..; M_k$ and $M_{l+1}; ..; M_k$ can be seen to be strictly decreasing by merging (if any) Bool modules (and redefining indices). Hence, the first module M_l in the subsequence of modules absorbs the attribute as

$$\text{gap\_att}_{|in\rangle,|out\rangle}(C) = |out\rangle \ominus C |in\rangle = (M_k \cdot \cdot M_l) C |in\rangle \ominus C |in\rangle \tag{8}$$
$$= Att_{|\psi\rangle}(M_k \cdots M_l) = Att_{|\psi\rangle}(M_l) \tag{9}$$

Similarly,

$$\text{gap\_att}_{|in\rangle,|out\rangle}(C; M_l) = |out\rangle \ominus C |in\rangle = (M_k \cdot \cdot M_{l+1}) M_l C |in\rangle \ominus M_l C |in\rangle \tag{10}$$
$$= Att_{M_l |\psi\rangle}(M_k \cdots M_{l+1}) = Att_{M_l |\psi\rangle}(M_{l+1}) \tag{11}$$

Since C^* is strictly decreasing, $Att_{|\psi\rangle}(M_l) > Att_{M_l |\psi\rangle}(M_{l+1})$. Replacing each by (8), (10), the criterion is satisfied.

Suppose $l - 1 = b - 2$. Then, $l + 1 = b$, so M_{l+1} is the first module of attribute Bool (while attribute of M_l is not Bool). Following the argument of the previous case,

$$\text{gap\_att}_{|in\rangle,|out\rangle}(C) = Att_{|\psi\rangle}(M_k \cdots M_l) = Att_{|\psi\rangle}(M_l) > \text{Bool}. \tag{12}$$

Since M_{l+1} is a Bool module and attribute of modules after M_{l+1} is Bool,

$$\text{gap\_att}_{|in\rangle,|out\rangle}(C; M_l) = Att_{M_l |\psi\rangle}(M_k \cdots M_{l+1}) \leq \text{Bool}, \tag{13}$$

satisfying the criterion as desired.

Suppose $l-1 > b-2$, which means that $l \geq b$. Then, $M_l, M_{l+1}, ..$ are Bool modules. Hence

$$\text{gap\_att}_{|in\rangle,|out\rangle}(C) = Att_{|\psi\rangle}(M_k \cdots M_l) = Att_{|\psi\rangle}(M_l) = \textsc{Bool} \tag{14}$$

$$\text{gap\_att}_{|in\rangle,|out\rangle}(C; M_l) = Att_{M_l|\psi\rangle}(M_k \cdots M_{l+1}) \leq \textsc{Bool} \tag{15}$$

satisfying the criterion (note that $\text{gap\_att}_{|in\rangle,|out\rangle}(C)$ cannot be Id otherwise it contradicts to C^* being solution circuit). □

Note that Theorem 4.16 does not hold for the 'monotonic' decreasing case because $Att_{C|in\rangle}(M_l) = Att_{M_lC|in\rangle}(M_{l+1})$ could be the case.

By Theorem 4.16, when the solution circuit is strictly decreasing, our algorithm with the optimized version of pruning (is_gap_filled^{opt}) never misses the solution. The existence of strictly decreasing modularization is the core assumption of Theorem 4.16 and this assumption depends on how the set of candidate modules ($\mathcal{M}$) is prepared. In the trivial case, if $\mathcal{M}$ includes all the possible modules, then every circuit has a decreasing modularization (since the circuit itself is included as a module in $\mathcal{M}$). In practice, we observed that the assumption is still likely to hold; with our module-generation method in Section 4.4, for example, all single IO circuits considered in our evaluation have strictly decreasing modularizations (Section 5.3).

For is_gap_filled to be sound, the following two conditions need to be met:

(1) The solution circuit is monotonically decreasing:

$$Att_{|\psi_1\rangle}(M_1) \geq Att_{|\psi_2\rangle}(M_2) \geq .. \geq Att_{|\psi_k\rangle}(M_k)$$

(2) The head module of any consecutive subsequence absorbs the attribute of following modules:

$$Att_{M_{p-1} \cdot\cdot M_1|in\rangle}(M_{p+n} \cdots M_p) = Att_{M_{p-1} \cdot\cdot M_1|in\rangle}(M_p) \text{ for } 1 \leq p \leq (k-1), 1 \leq n \leq k-p \tag{16}$$

This condition is intended to exclude some redundant circuit constructions such that a module reverts the attribute constructed by previous modules. For example, the circuit of three modules $M_1 = H(q_0), M_2 = H(q_1), M_3 = H(q_0); H(q_1)$ is monotonically decreasing on input $|0\rangle$. However, $Att(M_3M_2M_1) = \textsc{Id}$ as M_3 reverts the superposition constructed by M_1 and M_2.

Theorem 4.17. *Let $E = \{(|in\rangle, |out\rangle)\}$ be an example and $C^* = M_1; \cdots ; M_k$ ($M_i \in \mathcal{M}$ and attribute of each M_i is not Identity) be the solution circuit to be synthesized such that $C^*(|in\rangle) = |out\rangle$. Suppose C^* is monotonically decreasing (by input $|in\rangle$). Further, assume the following holds:*

$$Att_{M_{p-1} \cdot\cdot M_1|in\rangle}(M_{p+n} \cdots M_p) = Att_{M_{p-1} \cdot\cdot M_1|in\rangle}(M_p) \text{ for } 1 \leq p \leq (k-1), 1 \leq n \leq k-p \tag{17}$$

Then, for any prefix $C = M_1; \cdots ; M_{l-1}$ ($l \leq k$) of C^*,

$$\text{is\_gap\_filled}(C, M_l, (|in\rangle, |out\rangle)) = True.$$

Proof. Let $|\psi\rangle = C|in\rangle = M_{l-1} \cdots M_1 |in\rangle$ be the output vector of C. By definition $M_k \cdots M_l |\psi\rangle = |out\rangle$ and thus

$$Att_{|\psi\rangle}(M_kM_{k-1} \cdots M_l) = |out\rangle \ominus |\psi\rangle = \text{gap\_att}_{|in\rangle,|out\rangle}(C).$$

Since C^* is decreasing and by the given condition of (17), $Att_{|\psi\rangle}(M_kM_{k-1} \cdots M_l) = Att_{|\psi\rangle}(M_l)$. Therefore, $\text{gap\_att}_{|in\rangle,|out\rangle}(C) = Att_{|\psi\rangle}(M_l)$, which satisfies the criterion. Note that if C^* is not decreasing, the condition (17) cannot hold. □

Our soundness theorems are naturally extended to the multi-IO case. Since the pruning for multi-IO problems is defined using the technique for the single IO case, the soundness condition for pruning by (6) is merely the existence of partially decreasing modularizations (that partially hold for some of E). In practice, we observed that this assumption is also likely to hold; all multi-IO benchmark problems in our evaluation had such modularizations.

4.4 Module Generation

In this section, we explain our procedure for generating modules, i.e., $\mathcal{M}_{C,E}$.

4.4.1 Naive Module Generation. The most straightforward method would be to build every module that might exist with a length up to k by exhaustively listing all the possible component gates in $\mathcal{G}$. This naive module-generation procedure is to define $\mathcal{M}_{C,E}$ regardless of C and E as follows:

$$\mathcal{M}_{C,E} = \mathcal{M}_k = \{G_1(\mathbf{q}_1); \cdots ; G_m(\mathbf{q}_m) \mid \mathbf{q}_i \subseteq \mathbf{q}, G_i \in \mathcal{G}, 1 \leq i \leq m, 1 \leq m \leq k\}.$$

However, this approach has performance problems because the number of modules produced using this method grows exponentially in k as follows:

$$|\mathcal{M}_k| = \sum_{j=1,\dots,k} |\{G(\mathbf{q}') \mid G \in \mathcal{G}, \mathbf{q}' \subseteq \mathbf{q}\}|^j \tag{18}$$

$$= \sum_{j=1,\dots,k} \left[\sum_{G_i \in \mathcal{G}} (P(N, g_i)) \right]^j \tag{19}$$

where g_i is the number of qubits for G_i and $P(a, b)$ denotes permutation (i.e, choosing b from a things). $P(N, g_i)$ is the number of gate operations possibly made by G_i on N-qubits. For example, the number of modules for a 3-qubit circuit with $k = 3$ and $\mathcal{G} = \{H, CNOT\}$ is

$$\sum_{j=1,2,3} |\{H(q_0), H(q_1), H(q_2), CNOT(q_0, q_1), CNOT(q_0, q_2),$$
$$CNOT(q_1, q_0), CNOT(q_1, q_2), CNOT(q_2, q_0), CNOT(q_2, q_1)\}|^j = 819.$$

This observation led us to develop a technique for effectively reducing the number of modules.

4.4.2 Our Approach. Our method dynamically selects the set of candidate modules during the synthesis algorithm that are likely to pass the pruning criterion. Let E be the given input-output specification and C be the current circuit during the algorithm. Given C and E, our goal is to find a subset $\mathcal{M}_{C,E} \subseteq \mathcal{M}_k$. To achieve this, we exclude modules that are unlikely to pass the criterion, is_gap_filled (i.e., is_gap_filled$(C, M, E) = False$). Note that a module M passes our criterion, is_gap_filled(C, M, E), if the module attribute coincides with the attribute gap between C and E:

$$Att_{C|in_i\rangle}(M) = \text{gap\_att}_{|in_i\rangle,|out_i\rangle}(C) \text{ for } (|in_i\rangle, |out_i\rangle) \in E.$$

Therefore, if modules could be categorized according to their attributes, we might avoid applying other types of modules unnecessarily. Let $\mathcal{M}_\omega$ be the set of modules whose attribute is ω. Then, we define $\mathcal{M}_{C,E}$ as follows:

$$\mathcal{M}_{C,E} = \bigcup_{(|in_i\rangle,|out_i\rangle)\in E} \mathcal{M}_{\text{gap\_att}_{|in_i\rangle,|out_i\rangle}(C)}. \tag{20}$$

Now, we explain how we define $\mathcal{M}_\omega$ for each $\omega \in \{\textsc{Entangle}, \textsc{Superposition}, \textsc{Phasing}, \textsc{Bool}\}$. Following Definition 4.5, we should ideally collect modules $M \in \mathcal{M}_\omega$ to be $Att_{|\psi\rangle}(M) = \omega$, where $|\psi\rangle$ is an input vector that will be used as input to modules M (i.e., $|\psi\rangle = C\,|in_i\rangle$ for some $(|in_i\rangle, |out_i\rangle) \in E$). Formally, we have the following "ideal" goal: given C and some input $|in_i\rangle$ in E, collect $\mathcal{M}_\omega$ as follows:

$$\mathcal{M}_\omega \overset{\text{goal}}{\approx} \{M \in \mathcal{M} \mid Att_{C|in_i\rangle}(M) = \omega\}.$$

However, it would be nontrivial to build modules $\mathcal{M}_\omega$ for every scenario where $|\psi\rangle$ could occur because $Att_{|\psi\rangle}(M) = \omega$ holds depending on the specified $|\psi\rangle$. Instead, we want M to have the attribute ω in a manner that is independent of the input:

$$\mathcal{M}_\omega \overset{\text{goal}}{\approx} \{M \in \mathcal{M} \mid Att(M) = \omega\}$$

where $Att(M)$ is defined as follows :

Definition 4.18 (Input independent Attribute). For a module M (or gate, unitary matrix etc. in general), its input-dependent attribute $Att(M)$ is one of the following : for some classical state $|x\rangle$

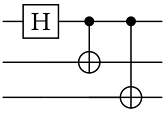
- Entangle if $M|x\rangle \ominus |x\rangle = \textsc{Entangle}$.
- Superposition if $M|x\rangle \ominus |x\rangle = \textsc{Superposition}$.
- Phasing if for another classical state $|y\rangle$, $M|x\rangle = e^{i\theta}|y\rangle$ for $\theta \in (0, 2\pi)$.
- Bool if $M|x\rangle$ is also classical state (strict to global phase).

When there is overlap in the categorization, we prioritize using Entangle > Superposition > Phasing > Bool.

We define $\mathcal{M}_\omega$ based on our observation of typical module patterns for each ω.

Preparation of $\mathcal{M}_{\textsc{Entangle}}$. Basically, module M that includes gate G of $Att(G) = \textsc{Entangle}$ will be $Att(M) = \textsc{Entangle}$. We observed another major pattern that superpositioning gates and inseparable gate operations are frequently applied in series to induce entanglement. We say a gate is inseparable if it cannot be factored out into tensor products of smaller unitary matrices V, W as $G = V \otimes W$ (e.g, $CNOT$ is inseparable since $CNOT = |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes X$).

For example, consider the following circuit that induces entanglement in `GHZ_from_100`:



The module maps unentangled state $|100\rangle$ to entangled one $\frac{1}{\sqrt{2}}(|000\rangle - |111\rangle)$. This entangling module is consistent with what we have observed: it is composed of a single Hadamard gate (H), which is a superpositioning gate, followed by a series of $CNOT$ gates that are inseparable.

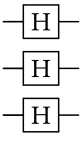
The detailed definition of this pattern is as follows. Assume the circuit size is N and the qubit register is $\mathbf{q} = \{q_1, .., q_N\}$. Assume for the moment we are creating entanglement from a state that is completely unentangled (i.e., the partition is $\{\{q_1\}, .., \{q_N\}\}$) to a state that is completely entangled (i.e., the partition is $\{\{\mathbf{q}\}\}$). Then, applying $N - 1$ inseparable gate operations will be sufficient if all inseparable gates are made of two qubits. Formally,

$$\begin{aligned}\mathcal{M}'_{\textsc{Entangle}} &= \{G_i^{sp}(\mathbf{q}_1^{sp}); \dots ; G_s^{sp}(\mathbf{q}_s^{sp}); G_1^{ins}(\mathbf{q}_1^{ins}); \dots ; G_{N-1}^{ins}(\mathbf{q}_{N-1}^{ins}) \\ &\qquad \mid Att(G_i^{sp}) = \textsc{Sp}, 1 \le s \le N, G_j^{ins} \text{ is inseparable}, \mathbf{q}_i^{sp}, \mathbf{q}_j^{ins} \subseteq \mathbf{q}\}\end{aligned}$$

If inseparable gates with more than two qubits are allowed as component gates, fewer inseparable sequences will be necessary. Summing up, the final definition for $\mathcal{M}_{\textsc{Entangle}}$ is defined as

$$\mathcal{M}_{\textsc{Entangle}} = \{G_1(\mathbf{q_1}); ..; G_s(\mathbf{q_s}) \mid \exists j. Att(G_j) = \textsc{Entangle}, 1 \le s \le k, 1 \le j \le s\} \cup \mathcal{M}'_{\textsc{Entangle}}$$

Preparation of $\mathcal{M}_{\textsc{Sp}}$. For $\mathcal{M}_{\textsc{Sp}}$, we collect modules that include at least one gate of attribute Sp (or Entangle gates since they also induce change in superposition). This is an obvious choice for $\mathcal{M}_{\textsc{Sp}}$; if module M is consisted only of Phasing and Bool, then its superposition status does not change and hence $Att(M) = \textsc{Phasing}$ or Bool. For example, the following is a typical pattern:



Formally, $\mathcal{M}_{\textsc{Sp}}$ is defined as follows:

$$\mathcal{M}_{\textsc{Sp}} = \left\{G_1(\mathbf{q}_1);\dots;G_s(\mathbf{q}_s) \middle| 1 \le s \le k, \exists j.Att(G_j) = \textsc{Sp}\right\}$$

Preparation of $\mathcal{M}_{\textsc{Phasing}}$ and $\mathcal{M}_{\textsc{Bool}}$. Basically, modules that only have Bool and Phasing gates with at least one Phasing gate would cause $Att(M) = \textsc{Phasing}$. Similarly, modules that only include Bool gates will satisfy $Att(M) = \textsc{Bool}$. These two cases are formally defined as follows

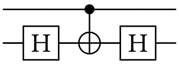
$$\mathcal{M}'_{\textsc{Phasing}} = \left\{ G_1(\mathbf{q}_1);\dots;G_s(\mathbf{q}_s) \;\middle|\; \begin{array}{l} Att(G_i) = \textsc{Phasing} \text{ or } \textsc{Bool} \text{ for } 1 \le i \le s, \\ \exists j \text{ s.t } Att(G_j) = \textsc{Phasing}, \\ 1 \le s \le k \end{array} \right\}$$
$$\mathcal{M}'_{\textsc{Bool}} = \left\{G_1(\mathbf{q}_1);\dots;G_s(\mathbf{q}_s) \;\middle|\; Att(G_j) = \textsc{Bool}, 1 \le j \le s, 1 \le s \le k\right\}$$

Additionally, modules of attributes Bool or Phasing are often generated by sandwiching them with two Sp gates. In this case, the front superposition gate sets up the input state to be represented in a transformed coordinate, and the end superposition gate restores the coordinate of the manipulated state.

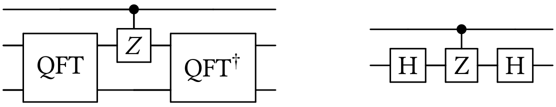
For example, consider gate operation $M = H(q_0);X(q_0);H(q_0)$, assuming input state vector $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. Hadamard gate transforms computational basis $|0\rangle$ and $|1\rangle$ to $|+\rangle$ and $|-\rangle$ respectively. Hence, applying (first) hadamard $H(q_0)$ on input state $|+\rangle$, it is represented in the transformed coordinate as $H(q_0)\,|+\rangle = |0\rangle$. Then, we manipulate it by $X(q_0)$ which becomes $|1\rangle$. By the end superposition gate, which is also hadamard $H(q_0)$ it restores the coordinate so that $|1\rangle = \frac{1}{\sqrt{2}}(|+\rangle - |-\rangle)$ becomes $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. Evaluating M, we can see that it was indeed identical to $Z(q_0)$ and hence $Att(M) = \textsc{Phasing}$. Modules of this pattern can be prescribed as follows:

$$\mathcal{M}^* = \left\{ G_1(\mathbf{q}_1);\dots;G_s(\mathbf{q}_s) \;\middle|\; \begin{array}{l} Att(G_1), Att(G_s) = \textsc{Superposition}, \mathbf{q}_1 = \mathbf{q}_s, \\ Att(G_j) = \textsc{Bool} \text{ or } \textsc{Phasing} \text{ and} \\ \mathbf{q}_j \cap \mathbf{q}_1 \neq \emptyset \text{ for } 2 \le j \le s-1 (\text{if } s \ge 3), \\ 2 \le s \le k \end{array} \right\}$$

For example, the following circuit has attribute Phasing:



and the following circuits are of attribute Bool:



Note that all modules consist of two Sp gates at the front and end, and some Bool, Phasing gates exist between them.

Summing up, the final definitions for $\mathcal{M}_{\textsc{Bool}}$ and $\mathcal{M}_{\textsc{Phasing}}$ are

$$\mathcal{M}_{\textsc{Phasing}} = \mathcal{M}'_{\textsc{Phasing}} \cup \mathcal{M}^*, \quad \mathcal{M}_{\textsc{Bool}} = \mathcal{M}'_{\textsc{Bool}} \cup \mathcal{M}^*.$$

Module Length. In our implementation, we generated modules of lengths up to $k = N$ (where N is the number of qubits), allowing more complex modules to be generated for circuits of larger qubit registers. As an exception, for $\mathcal{M}_{\textsc{Entangle}}$, we set $k = N + (N-1)$ (N for superposition and $(N-1)$ for the sequence of inseparables) to capture the patterns described by $\mathcal{M}'_{\textsc{Entangle}}$.

4.5 Complexity of Our Synthesis Algorithm

In this section, we analyze the time complexity of our synthesis algorithm (Algorithm 1). The runtime of the single iteration of the outer loop (lines 3–10 in Algorithm 1) is proportional to

$$|\mathcal{M}_k| \times (cost(\ominus) \times |E|)$$

where

- $\mathcal{M}_k$ is the set of modules given in (18), an upper bound of $\mathcal{M}_{C,E}$ used at line 5 of the algorithm,
- $|E|$ is the number of input-output examples (which is bounded as $|E| \leq 2^N$), and
- $cost(\ominus)$ denotes the cost of computing attribute difference (i.e, the $\ominus$ operator), which is invoked $|E|$ times when evaluating is_gap_filled at line 6.

Note that $\mathcal{M}_k$ depends on N (the number of qubits) and $\mathcal{G}$ (the set of component gates) as shown in (18). $cost(\ominus)$ is an expensive operation whose worst-case runtime is a double exponential function in the number of qubits (N). This is because it requires checking the existence of some permutation when computing the attribute difference for the Bool, Phasing, and Sp cases. However, since state vectors are mostly sparse, we observed the worst-case behavior is unlikely to occur in practice.

5 EVALUATION

In this section, we demonstrate the effectiveness of our synthesis algorithm.

5.1 Setup

To show the effectiveness, we evaluate and compare the following four variants of our algorithm:

- Base: A gate-level (not module-level) BFS-based enumerative synthesizer that prunes out identity gate sequences, $G_1(\mathbf{q}_1), G_2(\mathbf{q}_2), \ldots, G_n(\mathbf{q}_n)$ such that $\prod_{i=1}^{n} G_i(\mathbf{q}_i) = I$.
- Base$_{no\_prune}$: Base without pruning. This is included to check that Base is not very weak.
- Ours: The proposed module-level synthesizer with the optimized version of our pruning method (is_gap_filled^{opt} in Section 4).
- Ours$_{no\_prune}$: Ours without pruning (module-level search without pruning).

All experiments were carried out on an iMac with an Intel Core i5 processor (3.3 GHz).

Benchmarks. We collected 17 benchmark problems from various sources: online quantum programming exercises[1], textbook [Nielsen and Chuang 2011], online forum[2], and previous work on specific circuit construction [Draper 2000; Mastriani 2021; Neeley et al. 2010]. Table 1 shows the description of benchmarks, including input-output specifications and component gates used in experiments. Table 2 shows the solution circuits for each problem; to our knowledge, they are minimal in the number of provided gates.

5.2 Results

Table 3 reports the evaluation results. The results show that our module-level algorithm substantially improves the baseline, the gate-level synthesis method. Within a time limit of 3,600 seconds, Ours successfully synthesized 16 circuits out of 17, with an average synthesis time of 96.6 seconds. Among them, 11 circuits were synthesized in less than 10 seconds. Base, on the other hand, solved 10 out of 17 problems in an average time of 639.1 seconds. The overall speed-up for the 10 problems commonly solved by Ours and Base was 20.3x.

[1] https://github.com/microsoft/QuantumKatas

[2] https://quantumcomputing.stackexchange.com

Table 1. Benchmark problems used in evaluation, where $G_{\frac{p}{q}}$ denotes $\begin{pmatrix} \sqrt{p/q} & -\sqrt{1-(p/q)} \\ \sqrt{1-(p/q)} & \sqrt{(p/q)} \end{pmatrix}$ [Cruz et al. 2019].

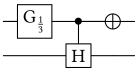
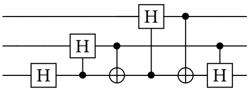
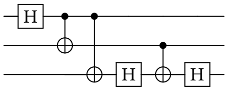
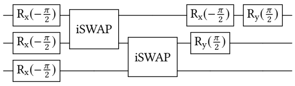
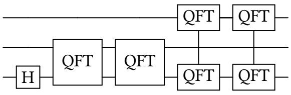
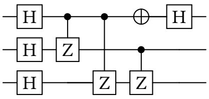
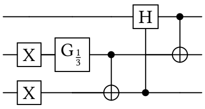
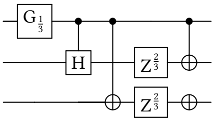
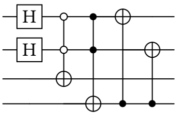
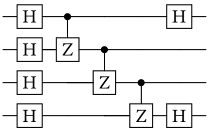
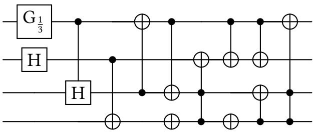
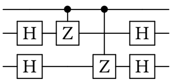
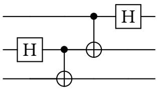
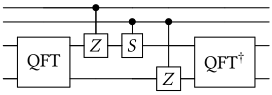
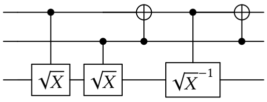
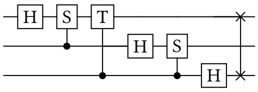
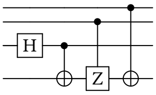
ID	Problem description and input-output specification	Component Gates
three_superpose	Prepare a 3-equal superposition of a 2-qubit state as $\lvert 00\rangle \mapsto \frac{1}{\sqrt{3}}(\lvert 00\rangle + \lvert 10\rangle + \lvert 01\rangle)$ [SE 2018a].	$G_{\frac{1}{3}}, CH, X$
M_valued	Prepare a state $\frac{1}{\sqrt{2}}(\lvert 0\rangle + \frac{1}{\sqrt{M}} \sum_{j=1}^{M} \lvert j\rangle)$ for $M = 4$ [SE 2018b].	$H, CH, CNOT$
GHZ_from_100	Prepare as GHZ, $\lvert 100\rangle \mapsto \lvert GHZ\rangle$, with restricted gates $H, CNOT$ [SE 2020c].	$H, CNOT$
GHZ_by_iSWAP	Prepare GHZ, $\lvert 000\rangle \mapsto \lvert GHZ\rangle$, using iSWAP and specific rotations [Neeley et al. 2010].	$iSWAP$, $R_x(-\frac{\pi}{2}), R_y(\frac{\pi}{2})$
GHZ_by_QFT	Prepare as $\lvert 000\rangle \mapsto \lvert GHZ\rangle$ specifically with QFT operations [Mastriani 2021] (note that 1-qubit QFT is H).	H, QFT
GHZ_Game	Prepare a state used in GHZ game as $\lvert 000\rangle \mapsto \frac{1}{2}(\lvert 000\rangle - \lvert 011\rangle - \lvert 101\rangle - \lvert 110\rangle)$ [QK 2022b].	H, X, CZ
W_orthog	Prepare 3-qubit state orthogonal to W-state [Dür et al. 2000] as : $\lvert 000\rangle \mapsto \frac{1}{\sqrt{3}}(\lvert 001\rangle - \lvert 010\rangle + \lvert 111\rangle)$ [SE 2020a].	$G_{\frac{1}{3}}, CH,$ $CNOT, X$
W_phased	Prepare 3-qubit W state with specific phases. 3-qubit version [QK 2022c]: $\lvert 000\rangle \mapsto \frac{1}{\sqrt{3}}(\lvert 001\rangle + \omega \lvert 010\rangle + \omega^2 \lvert 100\rangle)$ for $\omega = e^{\frac{2}{3}\pi i}$	$G_{\frac{1}{3}}, CH,$ $CNOT, Z^{\frac{2}{3}}, X$
W_four	Prepare 4-qubit W state: $\lvert 0000\rangle \mapsto \frac{1}{2}(\lvert 0001\rangle + \lvert 0010\rangle + \lvert 0100\rangle + \lvert 1000\rangle)$ [McClung 2020]	$H, NCNCNOT$ $TOFFOLI, CX$
cluster	Prepare a graph state [Hein et al. 2004]: $\lvert 0000\rangle \mapsto \frac{1}{2}(\lvert 0000\rangle + \lvert 0011\rangle + \lvert 1100\rangle - \lvert 1111\rangle)$. Inspired from question about understanding the circuit [SE 2020b].	H, CZ
bit_measure	Prepare a state of equal position, where the numbers of 1s in the first two bits and the last two bits are the same [SE 2018d]: $\lvert 0000\rangle \mapsto \frac{1}{\sqrt{6}}(\lvert 0000\rangle + \lvert 0101\rangle + \lvert 0110\rangle + \lvert 1001\rangle + \lvert 1010\rangle + \lvert 1111\rangle)$	$G_{\frac{1}{3}}, CH, X$ $CNOT, TOFFOLI$
flip	Implement bit flip code using H, CZ: for binary string state $\lvert b_1 b_2 b_3\rangle$ $(b_i \in \{0, 1\})$, if $b_1 = 1$, $\lvert 1 b_2 b_3\rangle \mapsto \lvert 1 \neg b_2 \neg b_3\rangle$, o.w identical map [SE 2019]	H, CZ
teleportation	Implement quantum teleportation. Motivated from question on StackExchange (see [SE 2018c] for provided input-output examples).	$H, CNOT$
indexed_bell	Indexed by first two qubits, prepare last two qubits to be in one of four Bell states [QK 2022a]: $\lvert 0000\rangle \mapsto \lvert 00\rangle \lvert \Phi^+\rangle, \lvert 0100\rangle \mapsto \lvert 01\rangle \lvert \Phi^-\rangle, \lvert 1000\rangle \mapsto \lvert 10\rangle \lvert \Psi^+\rangle, \lvert 1100\rangle \mapsto \lvert 11\rangle \lvert \Psi^-\rangle$	$H, CNOT, CZ$
toffoli_by_$\sqrt{X}$	Implement the Toffoli gate with controlled-$\sqrt{X}$ gates [Nielsen and Chuang 2011]: binary string state $\lvert b_1 b_2 b_3\rangle$, $\lvert 11 b_3\rangle \mapsto \lvert 11 \neg b_3\rangle$ o.w identical map	$C\sqrt{X}, C\sqrt{X}^{-1}, CNOT$
QFT	Implement Quantum Fourier Transformation(QFT) [Coppersmith 2002]: for 3-qubit binary state $\lvert x\rangle$, $\lvert x\rangle \mapsto \frac{1}{\sqrt{8}} \sum_{y=0}^{7} e^{\frac{2\pi i y x}{8}} \lvert y\rangle$	$H, CS, CT, SWAP$
draper	Implement Draper Adder [Draper 2000]: for 2-bit binary $a = a_2 a_1, b = b_2 b_1$, $\lvert b_2 b_1\rangle \lvert a_2 a_1\rangle \mapsto \lvert b_2 b_1\rangle \lvert (a+b) \bmod 4\rangle$	$CZ, CS,$ $QFT, QFT^{\dagger}$

The results also show that our module-level pruning contributed significantly to the speed up. To show the pruning's effectiveness, we compared Ours with Ours$_{no\_prune}$. Overall, Ours$_{no\_prune}$ succeeded to solve 14 out of 17 problems with an average time of 687.5 seconds. Ours, on the other hand, solved those 14 problems in an average time of 94.6 seconds (7.3x speed up).

5.3 Discussion

Modularization. We observed that quantum circuits often have a strictly decreasing modular decomposition. All the state preparation problems were solvable with such modularization, so that solution modules can be soundly captured by our pruning method, as Theorem 4.16 supports. For

Table 2. Solution circuits for the problems in Table 1. Circuits are in the (known) minimal forms.

Type	ID	Circuit	ID	Circuit
State Preparation	`three_superpose`		`M_valued`	
	`GHZ_from_100`		`GHZ_by_iSWAP`	
	`GHZ_by_QFT`		`GHZ_Game`	
	`W_orthog`		`W_phased`	
	`W_four`		`cluster`	
	`bit_measure`			
Multi IO	`flip`		`teleportation`	
	`draper`		`toffoli_by_`$\sqrt{X}$	
	`QFT`		`indexed_bell`	

example, `cluster` was synthesized by stacking modules in Figure 6e: M_1 is a module of attribute Entangle, M_2 is of attribute Superposition, and M_3 is of attribute Bool. By Proposition 4.11, we can check that this modularization is indeed strictly decreasing: Entangle > Superposition > Bool.

Another interesting case was the circuit that our algorithm found for `QFT` (Figure 6g). We note that it is not possible to find a proper modularization (that might pass (6)) by arbitrary slicing of `QFT` circuit in Table 2. However, our algorithm found that a proper modularization exists, as given in Figure 6g: for all inputs in the specification (E), M_1 fills the gap of Sp, M_2, M_3, M_4 sequentially fill the gap of Phasing, and the last M_5 changes in permutation, filling the gap of Bool. Also, note

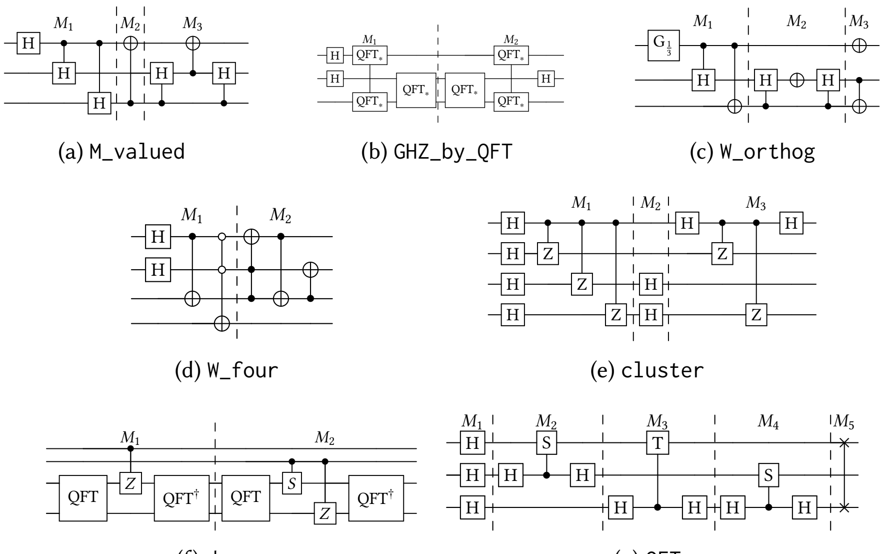
Table 3. Evaluation results. Synthesis time is given in seconds for each variant of our algorithm. ⊥ denotes time out (3600s). - denotes N/A. Spd-up : speed up in synthesis time of Ours over Base. #S : the number of gates in solution circuits at Table 2. #O: the number of gates in the circuit synthesized by Ours. Numbers in parentheses denotes the number of gates after post-processing (which applies trivial equality reduction). #M: the number of modules generated by Ours.

ID	Base$_{no\_prune}$	Base	Ours$_{no\_prune}$	Ours	Spd-up	#S	#O	#M
`three_superpose`	0.14	0.12	0.12	0.09	1x	3	3	2
`M_valued`	1764.75	1126.79	666.25	3.89	290x	6	7	3
`GHZ_from_100`	106.81	48.16	⊥	0.47	102x	6	6	2
`GHZ_by_iSWAP`	⊥	⊥	690.67	2.19	-	8	8	2
`GHZ_by_QFT`	116.90	101.65	101.17	39.26	3x	5	7	2
`GHZ_Game`	⊥	2305.71	4.51	0.57	4058x	8	8	2
`W_orthog`	2927.20	2075.06	248.23	2.43	854x	6	8	3
`W_phased`	⊥	⊥	258.56	5.43	-	7	7	3
`W_four`	⊥	⊥	2851.10	254.88	-	6	7	2
`cluster`	⊥	⊥	3560.38	8.91	-	9	13	3
`bit_measure`	⊥	⊥	⊥	⊥	-	13	-	-
`flip`	18.56	3.95	0.76	0.83	5x	6	6	2
`teleportation`	2.02	1.30	1.35	1.35	1x	4	4	2
`indexed_bell`	14.67	11.66	1.60	1.52	8x	4	4	2
`toffoli_by_`$\sqrt{X}$	956.29	716.28	306.10	264.66	3x	5	5	2
`QFT`	⊥	⊥	⊥	220.87	-	7	13(7)	5
`draper`	⊥	⊥	933.47	737.99	-	5	7(5)	2
Avg. (excluding ⊥/-)	656.37	639.07	687.45	96.58	20x			

that our modular representation involves more gate operations than the ideal solution (7 vs. 13), the number of gates can be easily reduced by applying known identities such as $H^2 = I$. After this post-processing, we obtain the same circuit presented in Table 2.

Similarly, the solution circuit for `draper` in Table 2 does not give a proper modularization, yet our algorithm could successfully find one as presented in Figure 6f. Also, note that the circuit in Figure 6f becomes equivalent to the solution circuit in Table 2 once we apply $QFT^{\dagger}QFT = I$.

Comparison with Compilation. As discussed in Section 1, our synthesizer aims to construct high-level circuits in terms of user-supplied component gates. On the other hand, existing compilers (e.g., qiskit-transpiler) produce low-level circuits in terms of a fixed set of primitive gate sets (e.g., $\{U_3, CNOT\}$). For example, consider the problem `cluster`. The circuit compiled by qiskit-transpiler is shown in Figure 7. Note that this circuit is difficult to read as it involves 43 gate operations and includes uninterpretable gate terms such as $U_3(\pi/2, 1.177, -\pi)$; programmers would not be able to understand how it works on input state vectors. Our synthesizer, on the other hand, produced a much smaller circuit in Figure 6e, when component set $\{H, CZ\}$ is given. `QFT` is another example. The circuit compiled by qiskit-transpiler is presented in Figure 8 and consists of 78 gates. By contrast, our algorithm found a circuit of size 7 (after post-processing) as presented in Figure 6g.



(a) M_valued (b) GHZ_by_QFT (c) W_orthog

(d) W_four (e) cluster

(f) draper (g) QFT

Fig. 6. Circuit synthesized by Ours. Shown only when they differ from solutions in Table 2. Dotted lines marks stacked modules in circuit synthesis. QFT_* denotes application of QFT in reversed qubit order (i.e, $QFT_*(a, b) = QFT(b, a)$).

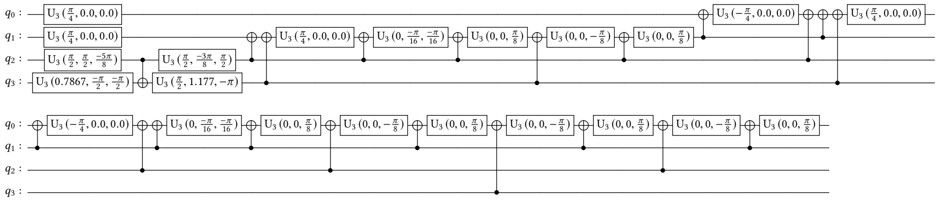


Fig. 7. Quantum circuit for cluster generated by qiskit-transpiler

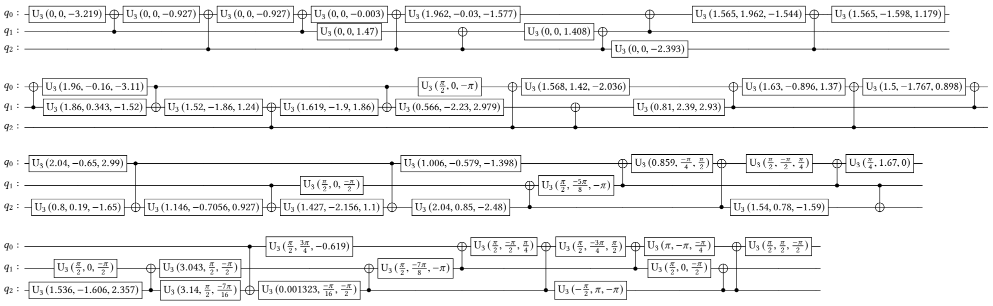


Fig. 8. Quantum circuit for QFT generated by qiskit-transpiler

5.4 Limitations and Future Work

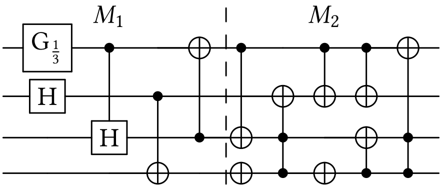
During the evaluation, we identified limitations and future work as follows.

Difficulty of Inferring Component Gates. To use our method, users need to provide component gates suitable for each synthesis problem, which can be nontrivial in general. For example, without any prior knowledge, it is nontrivial to infer $\{H, CS, CT, SWAP\}$ for `QFT`.

According to our experience, however, providing the right set of components was frequently not difficult and much easier than the synthesis problem itself. For example, component gates were sometimes already specified in the problem description itself, e.g., `GHZ_from_100` ("Prepare the GHZ state using H and CNOT?"), and in this case, no additional effort for inferring components was needed. Also, it is trivial to infer components for the problem `GHZ_by_QFT` ("Prepare GHZ using QFT"), which directly specifies the component gate (QFT).

Even when component gates are not given a priori, we could sometimes assume proper component gates without much difficulty. For example, consider the problem `W_phased`, where the input-output specification is given as $|000\rangle \mapsto \frac{1}{\sqrt{3}}(|001\rangle + \omega\,|010\rangle + \omega^2\,|100\rangle)$ (where let $\omega = e^{\frac{2}{3}\pi i}$). The required component gates in this case are $\{G_{\frac{1}{3}}, CH, CNOT, X, Z^{\frac{2}{3}}\}$. Inferring the gate $G_{\frac{1}{3}}$ from the input-output specification is not difficult because the specification says that we must introduce the amplitude $\frac{1}{\sqrt{3}}$: $G_{\frac{1}{3}}\,|0\rangle = \frac{1}{\sqrt{3}}\,|0\rangle + \sqrt{\frac{2}{3}}\,|1\rangle$. Deducing $Z^{\frac{2}{3}}$ is also possible because it is the phasing operation for ω: $Z^{\frac{2}{3}}(|0\rangle + |1\rangle) = (|0\rangle + \omega\,|1\rangle)$. The choice of $\{CNOT, X\}$ is common as they are frequently used to implement permutations. Also, CH is a well-known gate to introduce superposition. However, note that finding the solution circuit remains still challenging even when we know these components in advance.

Scalability. Our modular method failed in synthesizing `bit_measure` within 3,600 seconds. One possible modular representation of `bit_measure` that our method could find is the following:



where modules M_1 and M_2 have attributes Entangle and Bool, respectively. However, in order to find such a solution, the set $\mathcal{M}$ of candidate modules should include substantially large and complex modules with size 5 and various inseparable gates (CH, $CNOT$, $TOFFOLI$), which increases the number of candidate modules substantially. To address this issue, a more effective method for module generation would be required.

Circuits without Decreasing Modularization. The "decreasing modularization" property, which is the core condition for our synthesis algorithm to be sound, depends on how the set of modules ($\mathcal{M}_\omega$) is configured. So, the circuits that are not amenable to our approach are characterized by the set $\mathcal{M}_\omega$. For example, when $\mathcal{M}_\omega$ contains all the possible modules, all circuits have the "decreasing modularization" property and are amenable to our method. When $\mathcal{M}_\omega$ does not contain all modules, like ours used in the evaluation, some circuits may not have the property. Although our choice for $\mathcal{M}_\omega$ was expressive enough to capture the desired strict decreasing modularizations for benchmarks in Table 1, there exist exceptional cases. For example, consider the problem of synthesizing a circuit satisfying $|00\rangle \mapsto |+\rangle\,|1\rangle$ with component gates $\mathcal{G} = \{H, X^{1/3}\}$. In this case, the solution circuit will be

$$H(q_0); X^{1/3}(q_1); X^{1/3}(q_1); X^{1/3}(q_1).$$

Ideally, the solution circuit may have decreasing modularization by two modules; $M_1 = H(q_0)$ of attribute Sp and $M_2 = X^{1/3}(q_1); X^{1/3}(q_1); X^{1/3}(q_1)$ of attribute Bool. However, by our definition of $\mathcal{M}_{\text{Bool}}$, M_2 cannot be captured, hence the solution circuit does not satisfy the decreasing modularization property. As future work, to handle this issue a more thorough definition for each $\mathcal{M}_\omega$ should be considered (for example, considering $\mathcal{G}$-adaptive definition for $\mathcal{M}_\omega$).

6 RELATED WORKS

Quantum Circuit Synthesis. A large amount of research has been conducted on quantum circuit synthesis, known as unitary synthesis [Davis et al. 2019, 2020; Goubault de Brugière et al. 2020; Iten et al. 2016; Möttönen et al. 2004; Ross 2015; Shende et al. 2006; Smith et al. 2022; Tucci 2005; Younis et al. 2020]. One major approach is matrix decomposition, which recursively decomposes the provided unitary matrix into smaller matrices until they reach primitive gates. For example, Cartan's KAK decomposition [Tucci 2005] factorizes a 2-qubit unitary $U \in SU(4)$ into a non-local unitary matrix sandwiched by several local unitary matrices. More generally, Shende et al. [2006] introduced Quantum Shanon Decomposition, which is popular for reducing the $CNOT$ counts. Most of these matrix decomposition methods are complete: for any unitary matrix U, they are guaranteed to synthesize a circuit implementing U with an arbitrarily small error ϵ. Another approach to unitary synthesis is to parameterize circuits over (continuous) parameters of gates and then use numerical optimization [Davis et al. 2019, 2020; Smith et al. 2022; Younis et al. 2020].

Existing work on unitary synthesis differs from ours. The goal of unitary synthesis is to generate circuits with a fixed set of low-level gates (e.g., $CNOT + U_3$). In contrast, our method works for any discrete gate sets including high-level ones such as QFT and various controlled operations. Also, circuits resulting from unitary synthesis methods often suffer from large sizes (the number of gates, e.g., Figure 8), low fidelity, etc.

Recently, Paradis et al. [2021] presented a technique for automatically synthesizing uncomputation routines. This is also a different problem from ours; we aim to generate a quantum circuit in general that satisfies user-provided input-output specification while the focus of Paradis et al. [2021] is on synthesizing task-specific routines for uncomputation.

Program Synthesis. Our technique is an example of enumerative program synthesizers [Feser et al. 2015; Lee 2021; Osera and Zdancewic 2015; So and Oh 2017; Udupa et al. 2013; Wang et al. 2017], which have been used to synthesize SQL queries [Wang et al. 2017], functional data transformers [Feser et al. 2015], distributed protocols [Udupa et al. 2013], imperative programs [So and Oh 2017], etc. These techniques are typically used with search-space pruning based on, for example, abstract interpretation [So and Oh 2017; Wang et al. 2017], deduction [Feser et al. 2015], and types [Osera and Zdancewic 2015]. In this paper, we present domain-specific search and pruning techniques for synthesizing quantum circuits.

Component-based synthesis is also well-known in the literature, e.g., synthesizing bit-vector programs [Jha et al. 2010] API sequences [Feng et al. 2017b], and data wrangling scripts [Feng et al. 2017a]. In this paper, we apply the method to the domain of quantum circuits.

Quantum Programming Languages. As abstractions of quantum circuits, several quantum programming languages have been released [Aleksandrowicz et al. 2019; Bichsel et al. 2020; Developers 2022; Green et al. 2013; Svore et al. 2018; Yuan and Carbin 2022]. These languages provide high-level environments for quantum programming. As future work, our work could be extended to synthesize quantum programs in these languages.

7 CONCLUSION

Despite the enormous potential of quantum computers, writing quantum programs and algorithms is known to be notoriously difficult. In this paper, we suggested to address this problem with a method that automatically synthesizes quantum circuits from user-provided examples and component gates. We presented a quantum-specific algorithm for module-level enumerative search and pruning, and showed that this module-based approach is significantly faster than a gate-level synthesizer on a variety of synthesis tasks.

ACKNOWLEDGMENTS

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2020-0-01337,(SW STAR LAB) Research on Highly-Practical Automated Software Repair), the MSIT(Ministry of Science and ICT), Korea, under the ICT Creative Consilience program(IITP-2023-2020-0-01819) supervised by the IITP(Institute for Information & communications Technology Planning & Evaluation), and the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT)(No. 2021R1A5A1021944).

REFERENCES

Gadi Aleksandrowicz, Thomas Alexander, Panagiotis Barkoutsos, Luciano Bello, Yael Ben-Haim, David Bucher, Francisco Jose Cabrera-Hernández, Jorge Carballo-Franquis, Adrian Chen, Chun-Fu Chen, Jerry M. Chow, Antonio D. Córcoles-Gonzales, Abigail J. Cross, Andrew Cross, Juan Cruz-Benito, Chris Culver, Salvador De La Puente González, Enrique De La Torre, Delton Ding, Eugene Dumitrescu, Ivan Duran, Pieter Eendebak, Mark Everitt, Ismael Faro Sertage, Albert Frisch, Andreas Fuhrer, Jay Gambetta, Borja Godoy Gago, Juan Gomez-Mosquera, Donny Greenberg, Ikko Hamamura, Vojtech Havlicek, Joe Hellmers, Łukasz Herok, Hiroshi Horii, Shaohan Hu, Takashi Imamichi, Toshinari Itoko, Ali Javadi-Abhari, Naoki Kanazawa, Anton Karazeev, Kevin Krsulich, Peng Liu, Yang Luh, Yunho Maeng, Manoel Marques, Francisco Jose Martín-Fernández, Douglas T. McClure, David McKay, Srujan Meesala, Antonio Mezzacapo, Nikolaj Moll, Diego Moreda Rodríguez, Giacomo Nannicini, Paul Nation, Pauline Ollitrault, Lee James O'Riordan, Hanhee Paik, Jesús Pérez, Anna Phan, Marco Pistoia, Viktor Prutyanov, Max Reuter, Julia Rice, Abdón Rodríguez Davila, Raymond Harry Putra Rudy, Mingi Ryu, Ninad Sathaye, Chris Schnabel, Eddie Schoute, Kanav Setia, Yunong Shi, Adenilton Silva, Yukio Siraichi, Seyon Sivarajah, John A. Smolin, Mathias Soeken, Hitomi Takahashi, Ivano Tavernelli, Charles Taylor, Pete Taylour, Kenso Trabing, Matthew Treinish, Wes Turner, Desiree Vogt-Lee, Christophe Vuillot, Jonathan A. Wildstrom, Jessica Wilson, Erick Winston, Christopher Wood, Stephen Wood, Stefan Wörner, Ismail Yunus Akhalwaya, and Christa Zoufal. 2019. *Qiskit: An Open-source Framework for Quantum Computing*. https://doi.org/10.5281/zenodo.2562111

Benjamin Bichsel, Maximilian Baader, Timon Gehr, and Martin Vechev. 2020. Silq: A High-Level Quantum Language with Safe Uncomputation and Intuitive Semantics. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation* (London, UK) *(PLDI 2020)*. Association for Computing Machinery, New York, NY, USA, 286–300. https://doi.org/10.1145/3385412.3386007

D. Coppersmith. 2002. An approximate Fourier transform useful in quantum factoring. https://doi.org/10.48550/ARXIV.QUANT-PH/0201067

Diogo Cruz, Romain Fournier, Fabien Gremion, Alix Jeannerot, Kenichi Komagata, Tara Tosic, Jarla Thiesbrummel, Chun Lam Chan, Nicolas Macris, Marc-André Dupertuis, and Clément Javerzac-Galy. 2019. Efficient Quantum Algorithms for GHZ and W States, and Implementation on the IBM Quantum Computer. *Advanced Quantum Technologies* 2, 5-6 (2019), 1900015. https://doi.org/10.1002/qute.201900015 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/qute.201900015

Marc Grau Davis, Ethan Smith, Ana Tudor, Koushik Sen, Irfan Siddiqi, and Costin Iancu. 2019. Heuristics for Quantum Compiling with a Continuous Gate Set. https://doi.org/10.48550/ARXIV.1912.02727

Marc G. Davis, Ethan Smith, Ana Tudor, Koushik Sen, Irfan Siddiqi, and Costin Iancu. 2020. Towards Optimal Topology Aware Quantum Circuit Synthesis. In *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*. 223–234. https://doi.org/10.1109/QCE49297.2020.00036

Cirq Developers. 2022. *Cirq*. https://doi.org/10.5281/zenodo.6599601 See full list of authors on Github: https://github.com/quantumlib/Cirq/graphs/contributors.

Thomas G. Draper. 2000. Addition on a Quantum Computer. https://doi.org/10.48550/ARXIV.QUANT-PH/0008033

W. Dür, G. Vidal, and J. I. Cirac. 2000. Three qubits can be entangled in two inequivalent ways. *Phys. Rev. A* 62 (Nov 2000), 062314. Issue 6. https://doi.org/10.1103/PhysRevA.62.062314

Yu Feng, Ruben Martins, Jacob Van Geffen, Isil Dillig, and Swarat Chaudhuri. 2017a. Component-Based Synthesis of Table Consolidation and Transformation Tasks from Examples. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Barcelona, Spain) *(PLDI 2017)*. Association for Computing Machinery, New York, NY, USA, 422–436. https://doi.org/10.1145/3062341.3062351

Yu Feng, Ruben Martins, Yuepeng Wang, Isil Dillig, and Thomas W. Reps. 2017b. Component-Based Synthesis for Complex APIs. *SIGPLAN Not.* 52, 1 (jan 2017), 599–612. https://doi.org/10.1145/3093333.3009851

John K. Feser, Swarat Chaudhuri, and Isil Dillig. 2015. Synthesizing Data Structure Transformations from Input-Output Examples. *SIGPLAN Not.* 50, 6 (jun 2015), 229–239. https://doi.org/10.1145/2813885.2737977

András Gilyén, Yuan Su, Guang Hao Low, and Nathan Wiebe. 2019. Quantum Singular Value Transformation and beyond: Exponential Improvements for Quantum Matrix Arithmetics. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing* (Phoenix, AZ, USA) *(STOC 2019)*. Association for Computing Machinery, New York, NY, USA, 193–204. https://doi.org/10.1145/3313276.3316366

Timothée Goubault de Brugière, Marc Baboulin, Benoît Valiron, and Cyril Allouche. 2020. Quantum circuits synthesis using Householder transformations. *Computer Physics Communications* 248 (2020), 107001. https://doi.org/10.1016/j.cpc.2019.107001

Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoî t Valiron. 2013. Quipper. *ACM SIGPLAN Notices* 48, 6 (jun 2013), 333–342. https://doi.org/10.1145/2499370.2462177

Lov K. Grover. 1996. A Fast Quantum Mechanical Algorithm for Database Search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing* (Philadelphia, Pennsylvania, USA) *(STOC '96)*. Association for Computing Machinery, New York, NY, USA, 212–219. https://doi.org/10.1145/237814.237866

M. Hein, J. Eisert, and H. J. Briegel. 2004. Multiparty entanglement in graph states. *Physical Review A* 69, 6 (jun 2004). https://doi.org/10.1103/physreva.69.062311

Raban Iten, Roger Colbeck, Ivan Kukuljan, Jonathan Home, and Matthias Christandl. 2016. Quantum circuits for isometries. *Phys. Rev. A* 93 (Mar 2016), 032318. Issue 3. https://doi.org/10.1103/PhysRevA.93.032318

Susmit Jha, Sumit Gulwani, Sanjit A. Seshia, and Ashish Tiwari. 2010. Oracle-Guided Component-Based Program Synthesis. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1* (Cape Town, South Africa) *(ICSE '10)*. Association for Computing Machinery, New York, NY, USA, 215–224. https://doi.org/10.1145/1806799.1806833

Woosuk Lee. 2021. Combining the Top-down Propagation and Bottom-up Enumeration for Inductive Program Synthesis. *Proc. ACM Program. Lang.* 5, POPL, Article 54 (jan 2021), 28 pages. https://doi.org/10.1145/3434335

Guang Hao Low and Isaac L. Chuang. 2019. Hamiltonian Simulation by Qubitization. *Quantum* 3 (July 2019), 163. https://doi.org/10.22331/q-2019-07-12-163

Mario Mastriani. 2021. Quantum Fourier transform is the building block for creating entanglement. *Scientific Reports* 11, 1 (2021), 1–10.

James McClung. 2020. *Constructions and Applications of W-States*. Bachelor's Thesis. Worcester Polytechnic Institute.

Mikko Möttönen, Juha J. Vartiainen, Ville Bergholm, and Martti M. Salomaa. 2004. Quantum Circuits for General Multiqubit Gates. *Phys. Rev. Lett.* 93 (Sep 2004), 130502. Issue 13. https://doi.org/10.1103/PhysRevLett.93.130502

Matthew Neeley, Radoslaw C Bialczak, M Lenander, Erik Lucero, Matteo Mariantoni, AD O'connell, D Sank, H Wang, M Weides, J Wenner, et al. 2010. Generation of three-qubit entangled states using superconducting phase qubits. *Nature* 467, 7315 (2010), 570–573.

Michael A. Nielsen and Isaac L. Chuang. 2011. *Quantum Computation and Quantum Information: 10th Anniversary Edition* (10th ed.). Cambridge University Press, USA.

Peter-Michael Osera and Steve Zdancewic. 2015. Type-and-Example-Directed Program Synthesis. *SIGPLAN Not.* 50, 6 (jun 2015), 619–630. https://doi.org/10.1145/2813885.2738007

Anouk Paradis, Benjamin Bichsel, Samuel Steffen, and Martin Vechev. 2021. Unqomp: Synthesizing Uncomputation in Quantum Circuits. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation* (Virtual, Canada) *(PLDI 2021)*. Association for Computing Machinery, New York, NY, USA, 222–236. https://doi.org/10.1145/3453483.3454040

QK. 2022a. Task 1.7. https://github.com/microsoft/QuantumKatas/blob/main/Superposition/Workbook_Superposition.ipynb.

QK. 2022b. Task 2.1. https://github.com/microsoft/QuantumKatas/blob/main/GHZGame/Workbook_GHZGame.ipynb.

QK. 2022c. Task 2.3. https://github.com/microsoft/QuantumKatas/blob/0bc1b11ad2b29e358a9100dea7d0b9a973f5f9fd/Superposition/Workbook_Superposition_Part2.ipynb.

Neil J. Ross. 2015. Optimal Ancilla-Free CLIFFORD+V Approximation of Z-Rotations. *Quantum Info. Comput.* 15, 11–12 (sep 2015), 932–950.

SE. 2018a. How can I build a circuit to generate an equal superposition of 3 outcomes for 2 qubits? https://quantumcomputing.stackexchange.com/questions/2310/how-can-i-build-a-circuit-to-generate-an-equal-superposition-of-3-outcomes-for-2.

SE. 2018b. How can the state $|0\rangle + M^{-1/2} \sum_{j=1}^{M} |j\rangle$ be generated? https://quantumcomputing.stackexchange.com/questions/4545/how-can-the-state-lvert0-ranglem-1-2-sum-j-1m-lvert-j-rangle-be-genera.

SE. 2018c. How do I build a gate from a matrix on Qiskit? https://quantumcomputing.stackexchange.com/questions/4975/how-do-i-build-a-gate-from-a-matrix-on-qiskit.

SE. 2018d. How to create a quantum algorithm that produces $2n$-bit sequences with equal number of 1-bits? https://mathoverflow.net/questions/301733/how-to-create-a-quantum-algorithm-that-produces-2-n-bit-sequences-with-equal-num.

SE. 2019. Quantum circuit for a three-qubit bit-flip code. https://quantumcomputing.stackexchange.com/questions/9175/quantum-circuit-for-a-three-qubit-bit-flip-code.

SE. 2020a. Generalized construction of W basis. https://quantumcomputing.stackexchange.com/questions/13077/generalized-construction-of-w-basis.

SE. 2020b. Quantum Circuit explaination. https://quantumcomputing.stackexchange.com/questions/12552/quantum-circuit-explaination.

SE. 2020c. Transforming $|100\rangle$ state into $|000\rangle + |111\rangle$ state using only Hadamard and CNOT gates. https://quantumcomputing.stackexchange.com/questions/14642/transforming-100-rangle-state-into-000-rangle-111-rangle-state-using-on.

V.V. Shende, S.S. Bullock, and I.L. Markov. 2006. Synthesis of quantum-logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25, 6 (2006), 1000–1010. https://doi.org/10.1109/TCAD.2005.855930

Peter W. Shor. 1997. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM J. Comput.* 26, 5 (oct 1997), 1484–1509. https://doi.org/10.1137/S0097539795293172

Ethan Smith, Marc G. Davis, Jeffrey M. Larson, Ed Younis, Lindsay Bassman Oftelie, Wim Lavrijsen, and Costin Iancu. 2022. LEAP: Scaling Numerical Optimization Based Synthesis Using an Incremental Approach. *ACM Transactions on Quantum Computing* (aug 2022). https://doi.org/10.1145/3548693

Sunbeom So and Hakjoo Oh. 2017. Synthesizing Imperative Programs from Examples Guided by Static Analysis. In *Static Analysis*, Francesco Ranzato (Ed.). Springer International Publishing, Cham, 364–381.

Krysta Svore, Alan Geller, Matthias Troyer, John Azariah, Christopher Granade, Bettina Heim, Vadym Kliuchnikov, Mariia Mykhailova, Andres Paz, and Martin Roetteler. 2018. Q#: Enabling Scalable Quantum Computing and Development with a High-Level DSL. In *Proceedings of the Real World Domain Specific Languages Workshop 2018* (Vienna, Austria) *(RWDSL2018)*. Association for Computing Machinery, New York, NY, USA, Article 7, 10 pages. https://doi.org/10.1145/3183895.3183901

Qiskit Transpiler. 2022. transpiler-qiskit-transpiler. https://qiskit.org/documentation/apidoc/transpiler.html.

Robert R. Tucci. 2005. An Introduction to Cartan's KAK Decomposition for QC Programmers. https://doi.org/10.48550/ARXIV.QUANT-PH/0507171

Abhishek Udupa, Arun Raghavan, Jyotirmoy V. Deshmukh, Sela Mador-Haim, Milo M.K. Martin, and Rajeev Alur. 2013. TRANSIT: Specifying Protocols with Concolic Snippets. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Seattle, Washington, USA) *(PLDI '13)*. Association for Computing Machinery, New York, NY, USA, 287–296. https://doi.org/10.1145/2491956.2462174

Chenglong Wang, Alvin Cheung, and Rastislav Bodik. 2017. Synthesizing Highly Expressive SQL Queries from Input-Output Examples. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Barcelona, Spain) *(PLDI 2017)*. Association for Computing Machinery, New York, NY, USA, 452–466. https://doi.org/10.1145/3062341.3062365

Ed Younis, Koushik Sen, Katherine Yelick, and Costin Iancu. 2020. QFAST: Quantum Synthesis Using a Hierarchical Continuous Circuit Space. https://doi.org/10.48550/ARXIV.2003.04462

Charles Yuan and Michael Carbin. 2022. Tower: Data Structures in Quantum Superposition. *arXiv preprint arXiv:2205.10255* (2022).

Received 2022-10-28; accepted 2023-02-25