Jiang H, Li DK, Deng YX *et al.* A pattern matching-based framework for quantum circuit rewriting. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 33(1): 1–18 March 2024. DOI: 10.1007/s11390-024-2726-3

A Pattern Matching-Based Framework for Quantum Circuit Rewriting

Hui Jiang (蒋　慧), Dian-Kang Li (李典康), Yu-Xin Deng* (邓玉欣), *Distinguished Member*, *CCF*, Ming Xu (徐　鸣)

Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai 200062, China

E-mail: yhq_jh@126.com; 51205902048@stu.ecnu.edu.cn; yxdeng@sei.ecnu.edu.cn; mxu@cs.ecnu.edu.cn;

Received August 5, 2022; accepted March 19, 2024.

Abstract The realization of quantum algorithms relies on specific quantum compilations according to the underlying quantum processors. However, there are various ways to physically implement qubits and manipulate those qubits in different physical devices. These differences lead to different communication methods and connection topologies, with each vendor implementing its own set of primitive gates. Therefore, quantum circuits have to be rewritten or transformed in order to be transplanted from one platform to another. We propose a pattern matching-based framework for rewriting quantum circuits, called QRewriting. It takes advantage of a new representation of quantum circuits using symbolic sequences. Unlike the traditional approach using directed acyclic graphs, the new representation allows us to easily identify the patterns that appear non-consecutively but are reducible. Then, we convert the problem of pattern matching into that of finding distinct subsequences and propose a polynomial-time dynamic programming-based pattern matching and replacement algorithm. We develop a rule library for basic optimizations and rewrite the arithmetic and Toffoli circuits from a commonly used gate set to the gate set supported by the Surface-17 quantum processor. Compared with a state-of-the-art quantum circuit optimization framework PaF optimized on the BIGD benchmarks, QRewriting further reduces the depth and gate count by an average of 26.5% and 17.4%, respectively.

Keywords pattern matching, quantum circuit rewriting, subsequence

1 Introduction

Quantum computing has attracted more and more interest in the last decades since it provides the possibility to efficiently solve important problems such as integer factorization[1], unstructured search[2], and solving linear equations[3].

In recent years, with the popularity of quantum computing, many companies, universities, and institutes have been actively working to develop prototypes of quantum computers. For example, in 2019, Google announced the realization of quantum supremacy, the development of the 53-qubit quantum processor "Sycamore"[4]. In November 2021, IBM unveiled its new 127-qubit "Eagle" processor whose scale makes it impossible for a classical computer to reliably simulate, and the increased qubit count allows users to explore problems at a new level of complexity①. In June 2022, Xanadu demonstrated a quantum computational advantage with a programmable photonic processor that realized Gaussian boson sampling on 216 squeezed modes[5]. These systems are referred to as noisy intermediate-scale quantum systems[6] and have small qubit counts, restricted connectivity, and high gate error rates. The duration of a physical quantum

Regular Paper

This work was supported by the National Natural Science Foundation of China under Grant Nos. 62072176, 61832015, 12271172, and 11871221, the Shanghai Trusted Industry Internet Software Collaborative Innovation Center, and the "Digital Silk Road" Shanghai International Joint Lab of Trustworthy Intelligent Software under Grant No. 22510750100.

*Corresponding Author

①https://www.nature.com/articles/d41586-021-03476-5, Jun. 2022.

gate is roughly 10–800ns, if the fidelity of physical gates achieves at least 99%[7]. At present, the coherence time of each physical qubit is 1–150us and only a limited set of gates can be realized with relatively high fidelity on a quantum device. Each quantum processor may support a specific universal set of 1-qubit and 2-qubit gates, which are called primitive gates[7]. Table 1 lists three gate sets: G_{Com}, G_{IBM}, and G_{Sur}, where G_{Com} is a commonly used gate set[8], G_{IBM} is implemented by the IBM Q series[9], and G_{Sur} is used by the Surface-17 quantum processor[10].

Table 1. Three Gate Sets for Different Scenarios

Gate Set	Symbols
G_{Com}[8]	$\mathbf{H}, \mathbf{X}, \mathbf{Y}, \mathbf{Z}, \mathbf{S}, \mathbf{S}^\dagger, \mathbf{T}, \mathbf{T}^\dagger, \boldsymbol{R}_{\mathrm{z}}(\theta), \mathbf{CX}$
G_{IBM}[9]	$\boldsymbol{U}_1(\theta), \boldsymbol{U}_2(\alpha, \lambda), \boldsymbol{U}_3(\theta, \alpha, \lambda), \mathbf{CX}$
G_{Sur}[10]	$\mathbf{X}, \mathbf{Y}, \boldsymbol{R}_{\mathrm{x}}(\theta), \boldsymbol{R}_{\mathrm{y}}(\theta), \mathbf{CZ}$

The realization of quantum algorithms relies on specific quantum compilations, which mainly focus on the number of inserted quantum gates[11, 12] or the fidelity of the compiled quantum circuits[13, 14]. However, there are various ways to physically implement qubits in different physical devices and manipulate those qubits. These differences lead to different communication methods and connection topologies, with each vendor implementing its own set of primitive gates. Therefore, quantum circuits have to be transplanted from one platform to another. In addition, since the gate types supported by a quantum processor are limited, quantum circuits may also be rewritten when some high-level gates are decomposed into low-level gates before quantum circuits execute on the quantum processor.

Converting a quantum circuit supported by one gate set to a quantum circuit supported by another gate set with respect to some rules is called quantum circuit rewriting. Usually, a rule is in the form $\mathcal{C}_{\mathrm{p}} = \mathcal{C}_{\mathrm{s}}$, where $\mathcal{C}_{\mathrm{p}}$ is a fragment of a circuit whose behavior is the same as that of the fragment $\mathcal{C}_{\mathrm{s}}$. We call $\mathcal{C}_{\mathrm{p}}$ a pattern circuit and $\mathcal{C}_{\mathrm{s}}$ a substitution circuit. In this paper, we refer to the circuit to be rewritten as the target circuit. Motivated by the aforementioned requirements, our approach consists of two key steps: the first is to identify the patterns in the target circuit, the second is to replace them with semantically equivalent substitution circuits. For that purpose, we first introduce a new representation of quantum circuits using symbolic sequences. Unlike the traditional way of using directed acyclic graphs (DAGs) based on the execution dependencies of the circuit[15], the new representation allows us to easily identify the patterns that appear non-consecutively but are reducible. In the case that a fragment of a circuit can be matched by several different rules, we encounter a replacement conflict and need to resolve it with an appropriate policy. We propose three policies for generating schedulers to cope with replacement conflicts. One policy is precise in the sense that it will consider all the replacement candidates of a conflict set. In the worst case, its time complexity is exponential. For a large-scale circuit, we need to make a trade-off between the quality of the generated circuit and the time it takes. Therefore, for large-scale circuits, we propose a greedy policy to handle replacement conflicts and a stochastic policy to show that better policies exist.

The main contributions of this paper are summarized below.

- We introduce a new representation of quantum circuits, which can easily identify the patterns that appear non-consecutively but are reducible in the target circuits.
- We present a polynomial-time dynamic programming-based pattern matching and replacement algorithm.
- We propose three policies for generating schedulers to deal with replacement conflicts.
- We develop a rule library for basic optimizations.

The rest of the paper is structured as follows. Section 2 introduces the related work. Section 3 recalls the basic notations about quantum computing. Section 4

proposes a new representation of quantum circuits. Section 5 discusses the design of the pattern matching-based quantum circuit rewriting framework. Section 6 shows two case studies. Section 7 evaluates QRewriting by using the BIGD[16] benchmarks and a set of benchmark circuits[17] consisting of arithmetic circuits and implementations of multi-controlled Toffoli gates. Finally, Section 8 gives the conclusion.

2 Related Work

Several quantum circuit optimization compilers have recently been proposed to compile a quantum circuit to various processors. For example, Qiskit② and t|ket⟩[18] support generic gate sets, Quilc③ is tailored for the Rigetti Agave quantum processor. There are several optimizers that automatically discover patterns[19–22]. QRewriting aims to rewrite quantum circuits between different processors according to a given rule set, mainly focusing on pattern matching and replacement.

Pattern matching is widely used in circuit optimization. For example, many algorithms have employed peephole optimization and pattern matching to optimize circuits. Peephole optimization identifies small sets of gates and replaces them with equivalent sets that have better performance[13, 23]. Exact matching is only feasible for small-scale and medium-scale circuits[24]. Heuristics are often used in large-scale circuits, but they cannot ensure optimal results[25, 26]. Prasad *et al.* and Soekens *et al.* showed how to find optimal quantum circuits for all 3-qubit functions[27, 28]. Nam *et al.* proposed five optimization subroutines[17]. Murali *et al.* developed the first multi-vendor quantum computer compiler which compiles from high-level languages to multiple real-system quantum computer prototypes, with device-specific optimizations[9]. The work of Chen *et al.* is the closest to ours, where a quantum circuit optimization framework based on pattern matching (PaF) is proposed[29]. It uses subgraph isomorphism to find a pattern circuit in the target quantum circuit according to a given rule description, then replaces it with an equivalent one.

Previous work often treats a target circuit $\mathcal{C}$ as a DAG, which is usually not unique because various gates may commute. The patterns that appear non-consecutively but are reducible cannot be directly identified by subgraph isomorphism in the DAG. It can integrate with commutation analysis to adjust the order of commuting gates with time complexity $O(|\mathcal{C}|^2)$, where $|\mathcal{C}|$ is the length of the circuit $\mathcal{C}$.

In this paper, we introduce a new representation of quantum circuits, which can deal with non-consecutive patterns more conveniently. For quantum circuit rewriting, we propose a polynomial-time algorithm, which is based on dynamic programming to match and replace pattern circuits in the target circuit.

3 Preliminaries

In this section, we introduce some notions and notations of quantum computing. Let $\mathbb{Z}$ and $\mathbb{C}$ denote the sets of all integers and complex numbers, respectively.

Classical information is stored in bits, while quantum information is stored in qubits. Besides two basic states $|\mathbf{0}\rangle$ and $|\mathbf{1}\rangle$, a qubit can be in any linear superposition state $|\boldsymbol{\phi}\rangle = a\,|\mathbf{0}\rangle + b\,|\mathbf{1}\rangle$, where $a, b \in \mathbb{C}$ satisfy the condition $|a|^2 + |b|^2 = 1$. The state $|\boldsymbol{\phi}\rangle$ is observed to be in the state $|\mathbf{0}\rangle$ with probability $|a|^2$ and in the state $|\mathbf{1}\rangle$ with probability $|b|^2$.

A quantum gate acts on a collection of qubits, which are called the operation qubits of the gate. For example, the Hadamard ($\mathbf{H}$) gate is applied on one qubit, and the $\mathbf{CX}$ gate is applied on two qubits. Its behavior is described as

$$\mathbf{CX}(\alpha\,|\mathbf{0}\rangle\,|\boldsymbol{\psi}\rangle + \beta\,|\mathbf{1}\rangle\,|\boldsymbol{\phi}\rangle) = \alpha\,|\mathbf{0}\rangle\,|\boldsymbol{\psi}\rangle + \beta\,|\mathbf{1}\rangle\,(\mathbf{X}\,|\boldsymbol{\phi}\rangle).$$

That is, we apply an $\mathbf{X}$ gate to the second qubit (called the target) if the first (the control) is in the state $|\mathbf{1}\rangle$, and the identity transformation otherwise, where $|\boldsymbol{\psi}\rangle$ and $|\boldsymbol{\phi}\rangle$ are the state of the second qubit. Two

② https://github.com/Qiskit, Jun. 2022.

③ https://github.com/quil-lang/quilc, Jun. 2022.

other gates which are relevant include the 3-qubit Toffoli gate **CCX** and the double-controlled phase gate **CCZ**. Likewise, the **CCX** and **CCZ** gates apply **X** and **Z** gate, respectively, when the control qubits are in state $|\mathbf{1}\rangle$.

In a quantum circuit, each line represents a wire. The wire does not necessarily correspond to a physical wire, but may correspond to the passage of time or a physical particle that moves from one location to another through space. The interested reader can find more details of these gates in the standard textbook[8]. The execution order of a quantum logical circuit is from left to right. The width of a quantum circuit refers to the number of qubits in the quantum circuit. The depth of a quantum circuit refers to the number of layers of gates that are executable in parallel. We refer to a quantum circuit with a depth of less than 100 as a small-scale circuit, a quantum circuit with a depth of more than 1000 as a large-scale circuit, and the rest as medium-scale circuits.

4 Circuit Representation

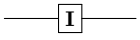
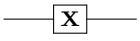
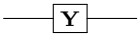
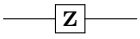
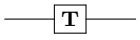
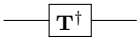
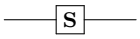
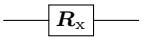
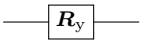
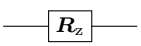
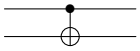
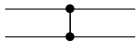
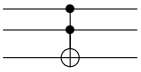
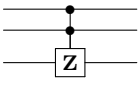
In this section, we define a new representation of quantum circuits and the pattern matching condition, which easily identifies the patterns that appear non-consecutively but are reducible. Based on that, we will state the quantum circuit rewriting problem considered in the paper.

Definition 1. *A gate is represented by a triple* (γ, σ, α)*, where*

- γ *is the symbol of a gate type,*
- σ *is a finite sequence of operation qubits for a gate,*
- α *is a finite sequence of rotation angles for a gate.*

A quantum circuit $\mathcal{C}$ is a sequence of triples $(\gamma_0, \sigma_0, \alpha_0)(\gamma_1, \sigma_1, \alpha_1)\cdots(\gamma_{n-1}, \sigma_{n-1}, \alpha_{n-1})$, and the length of the circuit is denoted by $|\mathcal{C}|$. The gate sequence $\Gamma_{\mathcal{C}}$ of the quantum circuit $\mathcal{C}$ is a symbolic sequence of gate types obtained by projecting each element of $\mathcal{C}$ to its first component, i.e., $\Gamma_{\mathcal{C}} =$ "$\gamma_0\gamma_1\cdots\gamma_{n-1}$". The new representation of a rule is a pair $r = (\mathcal{C}_{\rm p}, \mathcal{C}_{\rm s})$, consisting of a pattern circuit $\mathcal{C}_{\rm p}$ and a substitution circuit $\mathcal{C}_{\rm s}$. For simplicity, if the sequence is empty ϵ, we ignore it. Table 2 lists some of the commonly used quantum gates, their distinct aliases, and the corresponding circuit symbols.

Table 2. Symbols of Gates and Distinct Aliases

Gate	Aliase	Circuit
I	"I"	
H	"h"	
X	"x"	
Y	"y"	
Z	"z"	
T	"t"	
$\mathbf{T}^\dagger$	"T"	
S	"s"	
$\mathbf{S}^\dagger$	"S"	
$\boldsymbol{R}_{\rm x}$	"X'	
$\boldsymbol{R}_{\rm y}$	"Y"	
$\boldsymbol{R}_{\rm z}$	"Z"	
CX	"c"	
CZ	"C"	
CCX	"E"	
CCZ	"F"	

Example 1. We next illustrate the symbolic sequences of the circuit obtained by using the patterns in Fig.1 to rewrite the target circuit $\mathcal{C}_{\rm t}$ in Fig.2(a) as an example. The new representation of the quantum circuit is

$$\mathcal{C}_{\rm t} = (\text{"x"}, q_2)(\text{"x"}, q_2)(\text{"c"}, q_0q_1)(\text{"c"}, q_0q_2)(\text{"c"}, q_0q_1)$$
$$(\text{"x"}, q_2)(\text{"x"}, q_0),$$

and its gate sequence is represented by $\Gamma_{\rm t} =$ "xxcccxx". We can make use of the rule set $\mathcal{R} = \{r_0, r_1, r_2, r_3\}$ in Fig.1 to rewrite the circuit, where

$$r_0 = ((\text{"x"}, q_0)(\text{"x"}, q_0), (\text{"I"}, q_0)),$$

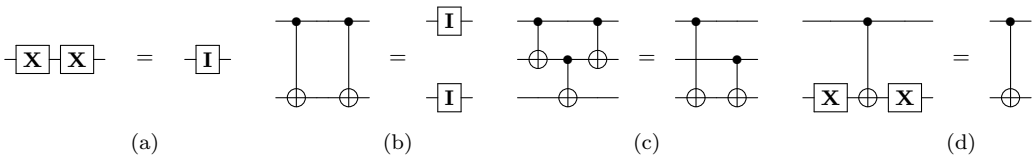


Fig.1. The rule set $\mathcal{R}$ used to optimize the X and CX gates. (a) r_0. (b) r_1. (c) r_2. (d) r_3.

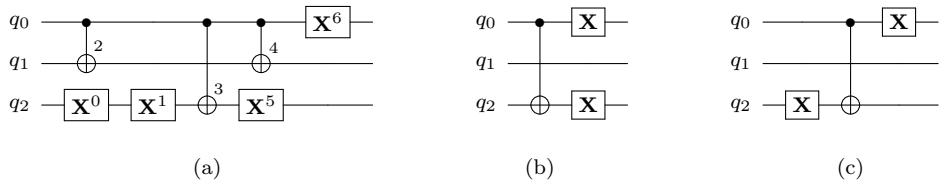


Fig.2. (a) A quantum circuit. The markers 0–6 on the gate symbols represent the order in the gate sequence. (b) and (c) are the results of the quantum circuit in (a) rewritten by schedulers s' and s'', respectively.

$$r_1 = ((\text{“c”}, q_0q_1)(\text{“c”}, q_0q_1), (\text{“I”}, q_0)(\text{“I”}, q_1)),$$

$$r_2 = ((\text{“c”}, q_0q_1)(\text{“c”}, q_1q_2)(\text{“c”}, q_0q_1), (\text{“c”}, q_0q_2)\ (\text{“c”}, q_1q_2)),$$

$$r_3 = ((\text{“x”}, q_1)(\text{“c”}, q_0q_1)(\text{“x”}, q_1), (\text{“c”}, q_0q_1)).$$

To facilitate the description of pattern matching, we introduce the following definitions.

Definition 2[30]. *Let Γ and Γ' be two sequences. We say Γ' is a subsequence of Γ, if there exist indices $0 \le i_0 < \cdots < i_{|\Gamma'|-1} < |\Gamma|$ such that $\Gamma[i_k] = \Gamma'[k]$ for all $k \in [0, |\Gamma'| - 1]$.*

The subsequence set is a set of distinct subsequences of the pattern circuit in the target circuit. We do not distinguish the indices from the gates to which the indices correspond in the quantum circuit. Suppose Γ is a gate sequence. Then $\Gamma[i : j]$ means to take the fragment of Γ from index i to j. If j is not specified, it takes the suffix of Γ from index i. Supposing that Γ' is a subsequence of Γ, we write $\Gamma \setminus \Gamma'$ to indicate the subsequence of Γ obtained by removing all occurrences of Γ'.

Example 2. Continuing to consider Example 1, the gate sequences and subsequence sets of the pattern circuits in the rule set $\mathcal{R}$ are given as follows:

- “xx”: $\{[0, 1]\}$,
- “cc”: $\{[2, 4]\}$,
- “ccc”: $\emptyset$,
- “xcx”: $\{[1, 3, 5]\}$.

Definition 3 (Qubit Mapping). *Given two qubit sets Q and Q', a qubit mapping function f is a bijective function between the qubit sets Q and Q'.*

Definition 4 (Qubit State Independence). *Let $\mathcal{C}_\mathrm{t}$, $\mathcal{C}_\mathrm{p}$ be two circuits with gate sequences Γ_t, Γ_p, respectively. Suppose a subsequence $\Gamma'_\mathrm{t} = \{i_0, \ldots, i_\mathrm{l}\}$ of Γ_t that can match Γ_p. We say the qubit state in Γ'_t is independent w.r.t. $\Gamma_\mathrm{t}[i_0 : i_\mathrm{l}] \setminus \Gamma'_\mathrm{t}$, if the control qubit set in Γ'_t does not intersect with the target qubit set of the gates in $\Gamma_\mathrm{t}[i_0 : i_\mathrm{l}] \setminus \Gamma'_\mathrm{t}$, and vice-versa.*

Definition 5 (Pattern Matching). *Let $\mathcal{C}_\mathrm{t}$ and $\mathcal{C}_\mathrm{p}$ be a target circuit and a pattern circuit with gate sequences Γ_t, Γ_p, respectively. We say $\mathcal{C}_\mathrm{p}$ matches $\mathcal{C}_\mathrm{t}$ if the following two conditions hold:*

- *Γ_t has a subsequence that can match Γ_p up to a qubit mapping function.*
- *The qubit sets of the subsequence and the pattern circuit satisfy the qubit mapping function and the qubit state independence condition.*

Example 3. We continue the Example 2 to show the difference between the new representation of quantum circuits and the DAG representation. Suppose the gates of the target circuit $\mathcal{C}_\mathrm{t}$ (resp. pattern circuit of r_1) from left to right are named g_0–g_6 (resp. g'_0–g'_1). Fig.3(a) is the DAG representation of the circuit fragment g_2–g_4

and Fig.3(c) is the DAG representation of the pattern circuit of r_1. We can see that Fig.3(a) has no subgraph isomorphic to Fig.3(c). The gates g_2 and g_3 in Fig.3(a) are exchanged by commutation analysis to obtain Fig.3(b), which contains a subgraph isomorphic to Fig.3(c). The symbolic sequence of the circuit $\mathcal{C}_\mathrm{t}$ can directly match the pattern circuit of r_1, i.e., the gates g_2 and g_4. It satisfies the qubit mapping function $\{f(q_0) = q_0,\ f(q_1) = q_1\}$ and the qubit state independence condition. Therefore, the circuit $\mathcal{C}_\mathrm{t}$ can be rewritten as $(\text{“x”}, q_2)(\text{“x”}, q_2)(\text{“c”}, q_0q_2)(\text{“x”}, q_2)(\text{“x”}, q_0)$.

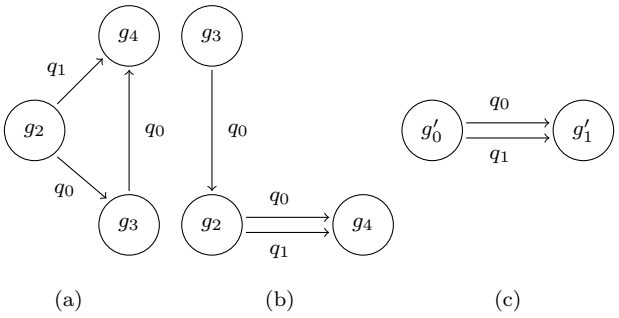


Fig.3. (a) DAG representation of the circuit fragment g_2–g_4 of $\mathcal{C}_\mathrm{t}$. (b) DAG representation of the circuit fragment g_3, g_2, g_4 by exchanging the gates g_2 and g_3 of (a). (c) Pattern circuit of the rule r_1.

Definition 6. *For a given target circuit $\mathcal{C}_\mathrm{t}$ and a rule $r = (\mathcal{C}_\mathrm{p}, \mathcal{C}_\mathrm{s})$, a replacement candidate is a triple (D, r, e), where*

- *D is a subsequence of the target circuit $\mathcal{C}_\mathrm{t}$ that can match the pattern circuit $\mathcal{C}_\mathrm{p}$,*
- *r is a rule,*
- *$e \in \mathbb{Z}$ is a conflict index, with the default value being -1.*

Definition 7. *A target circuit has a replacement conflict if an index of the target circuit appears more than once in the subsequence set.*

The replacement candidates for a replacement conflict form a conflict set. A replacement scheduler is a set of replacement candidates for different indices.

Example 4. Continuing to consider Example 3, we see that the subsequence set of the sequence “cc” is $\{[2, 4]\}$. The subsequence $[2, 4]$ appears non-consecutively in the sequence Γ_t. The gate $(\text{“x”}, q_2)$ appears in both the subsequence sets of “xx” and “xcx”, which means that in the target circuit, different rules may be matched at the same index. Two schedulers are given as follows:

$$s' = \{([0, 1], r_0, 1), ([2, 4], r_1)\},$$
$$s'' = \{([1, 3, 5], r_3, 1), ([2, 4], r_1)\}.$$

The scheduler s' (resp. s'') replaces the gates in the sequence $[0, 1]$ (resp. $[1, 3, 5]$) using the substitution circuit of r_0 (resp. r_3). After one of the schedulers is applied, we get the circuit in Fig.2(b) or Fig.2(c). Different schedulers result in different gate counts or depths of the rewritten circuits. The circuits rewritten by the scheduler s' or s'' have the same gate counts but with depths are 2 and 3, respectively. In both schedulers, the first element has the component 1, which is an index to indicate where the conflict takes place.

We are now ready to state the following problem.

Problem 1. Given two gate sets G_1, G_2 and a rule set $\mathcal{R}$ that expresses the equivalence of each gate in G_1 in terms of elements of G_2, how can one rewrite a quantum circuit supported by a gate set G_1 to a quantum circuit supported by G_2?

By using the new representation of quantum circuits, we reduce the above problem to find distinct subsequences of the pattern sequence in the target sequence up to a qubit mapping function and we use a qubit state independence condition to filter the obtained subsequences.

5 Quantum Circuit Rewriting

We propose a pattern matching-based quantum circuit rewriting framework. It consists of two steps. One matches the pattern circuit in the target circuit, the other replaces it.

5.1 Pattern Matching Algorithm

We propose an algorithm based on dynamic programming to match the patterns in a rule set against a target circuit. Let $\mathcal{C}_\mathrm{t}$ and $\mathcal{C}_\mathrm{p}$ be the target and pattern circuits with gate sequences Γ_t and Γ_p, respectively.

We consider the problem of finding the distinct subsequences of the pattern sequence Γ_p in the target sequence Γ_t. The obtained subsequences only match the gate types, thus we need to check whether the operation qubits in the subsequences satisfy the qubit mapping function and the qubit state independence condition.

The input of Algorithm 1 is a target circuit $\mathcal{C}_\mathrm{t}$ and a rule set $\mathcal{R}$, and the output is a set of replacement candidates $\mathcal{M}$. The function subseq(Γ_t, Γ_p, δ) in Algorithm 2[31] uses a dynamic programming algorithm to compute the distinct subsequences of Γ_t that can match Γ_p and returns the subsequence set to a set D. The function check_qubit_condition(D, $\mathcal{C}_\mathrm{t}$, r) checks whether the subsequences in D satisfy the qubit mapping function and the qubit state independence condition.

The input of the function subseq(Γ_t, Γ_p, δ) is a target sequence Γ_t, a pattern sequence Γ_p, and a parameter δ to limit the range of indices of Γ_p in Γ_t. The output is a set $D[|\Gamma_\mathrm{p}|+1]$ recording the distinct subsequences of Γ_t that can match Γ_p. We use the set $D[j+1]$ to record the subsequences of Γ_t that can match $\Gamma_\mathrm{p}[0:j]$. If the condition $\Gamma_\mathrm{t}[i]=\Gamma_\mathrm{p}[j]$ is satisfied, there are subsequences of $\Gamma_\mathrm{t}[0:i]$ that can match $\Gamma_\mathrm{p}[0:j]$. Line 6 updates the set $D[j+1]$. To find the subsequence of $\Gamma_\mathrm{p}[0:j]$, we need to first calculate the subsequences of $\Gamma_\mathrm{p}[0:j-1]$. The update $D[j+1] \leftarrow D[j+1] \cup \{u.\mathrm{append}(i) : u \in D[j] \text{ and } i-u[0]<\delta\}$ is the Bellman equation[31], which is a necessary enabler of the dynamic programming algorithm. The time complexity of Algorithm 1 is $O(|\mathcal{R}| \times |\Gamma_\mathrm{p}| \times |\Gamma_\mathrm{t}|)$ and the space complexity is $O(|\Gamma_\mathrm{p}|)$.

Algorithm 1: pattern_matching($\mathcal{C}_\mathrm{t}$, $\mathcal{R}$)

Input: a quantum circuit $\mathcal{C}_\mathrm{t}$ and a rule set $\mathcal{R}$;
Output: a set of substitution candidate $\mathcal{M}$;
1 $\mathcal{M} \leftarrow \emptyset$;
2 let Γ_t be the gate sequence of circuit $\mathcal{C}_\mathrm{t}$;
3 **for** $r \in \mathcal{R}$ **do**
4 let Γ_p be the gate sequence of pattern circuit of r;
5 $D \leftarrow$ subseq(Γ_t, Γ_p, δ);
6 **if** check_qubit_condition(D, $\mathcal{C}_\mathrm{t}$, r) **then**
7 $\mathcal{M} \leftarrow \mathcal{M} \cup \{(u, r, -1) : u \in D\}$;
8 **return** $\mathcal{M}$;

Algorithm 2: subseq(Γ_t, Γ_p, δ)

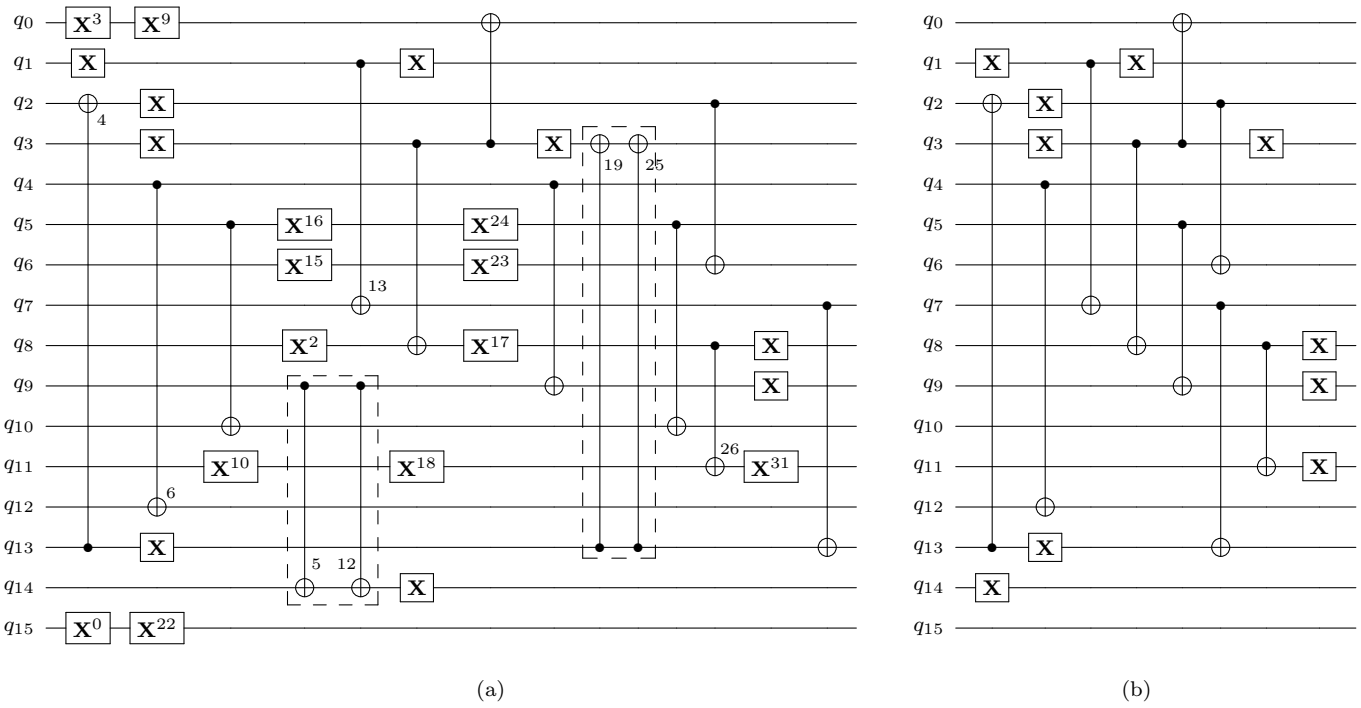
Input: two sequences Γ_t and Γ_p, and a parameter δ;
Output: a set of subsequences of Γ_p;
1 let D be a sequence of length $|\Gamma_\mathrm{p}|+1$ and each element is an empty set;
2 let ϵ be an empty sequence and $D[0] \leftarrow \{\epsilon\}$;
3 **for** $i \leftarrow 0$ to $|\Gamma_\mathrm{t}|$ **do**
4 **for** $j \leftarrow \min(i, |\Gamma_\mathrm{p}|)$ to 0 **do**
5 **if** $\Gamma_\mathrm{t}[i] = \Gamma_\mathrm{p}[j]$ **then**
6 $D[j+1] \leftarrow D[j+1] \cup \{u.\mathrm{append}(i) : u \in D[j] \text{ and } i-u[0]<\delta\}$;
7 **return** $D[|\Gamma_\mathrm{p}|+1]$;

Example 5. Let us consider the quantum circuit in Fig.4(a) and the rule set $\mathcal{R} = \{r_0, r_1, r_2, r_3\}$ in Fig.1. The gate sequences of the target circuit and the pattern circuits are "xxxxcccxxxxxccxxxxcccxxxcccxxxxxxxccc", "xx", "cc", "ccc", and "xcx", respectively. The subsequence set of "cc" is $\{[4,5],[4,6],[5,6],\cdots\}$. The corresponding gates of the indices $\{4,5,6\}$ in the target circuit $\mathcal{C}_\mathrm{t}$ are ("c", $q_{13}q_2$)("c", q_9q_{14})("c", q_4q_{12}). The subsequences $[4,5]$, $[4,6]$, $[5,6]$ do not satisfy the qubit mapping function condition. The function check_qubit_condition(D, $\mathcal{C}_\mathrm{t}$, r) filters the subsequences and finally gets the subsequence set of "cc" in the gate sequence of the target circuit, which is $\{[5,12],[19,25]\}$ highlighted with dotted lines in Fig.4(a).

5.2 Replacement Algorithm

By Algorithm 1, we obtain all the subsequences of the pattern circuits in the target circuit. To resolve replacement conflicts, we propose three conflict resolution policies. They give rise to three variants of QRewriting called QPRewriting, QGRewriting, and QSRewriting, respectively. Note that QRewriting uses the greedy policy by default. Due to the decoherence of qubits, the lifetime of qubits is very short[32]. The execution time of a quantum circuit is determined by several factors such as the depth and the gate count of the quantum circuit. Here, we mainly use depth to select the optimal replacement scheduler.

- The precise policy calculates all the candidates



(a) (b)

Fig.4. (a) The quantum circuit 16QBT_05YCTFL_3. The markers 0–31 on the gate symbols represent the order in the gate sequence. (b) The quantum circuit 16QBT_05YCTFL_3 optimized by QSRewriting.

when a replacement conflict occurs.

- The greedy policy chooses the candidate that appears first in the target circuit among the conflict set.
- The stochastic policy selects a candidate stochastically in the conflict set for the scheduler.

We propose an algorithm based on the breadth-first search to compute the replacement scheduler as shown in Algorithm 3. The input is a gate sequence $\Gamma_{\rm t}$ and a set of replacement candidates $\mathcal{M}$. $\mathcal{S}$ is a scheduler set, and the queue $\mathcal{S}_{\rm q}$ stores the sub-scheduler. Firstly, we push an empty scheduler $\emptyset$ into the queue $\mathcal{S}_{\rm q}$. Then, we loop the queue $\mathcal{S}_{\rm q}$, lines 4–12, until it is empty. The function next_conflict($\Gamma_{\rm t}$, $\mathcal{M}$, s) computes the next conflict index e in $\Gamma_{\rm t}$ from the current conflict index to the end of $\Gamma_{\rm t}$. If there is no conflict at this index, we directly add the replacement candidate to s. Otherwise, we return the index. When arriving at the end of $\Gamma_{\rm t}$, we add the scheduler s into $\mathcal{S}$. Line 10, according to the conflict policy, calculates the candidate set that has a conflict at index e in $\mathcal{M}$. Lines 11 and 12 append the replacement candidates to s and push it into queue $\mathcal{S}_{\rm q}$. Finally, we calculate the depth of the replaced circuit and return the scheduler with the smallest depth.

Algorithm 3: solve_conflicts($\Gamma_{\rm t}$, $\mathcal{M}$)

Input: a gate sequence $\Gamma_{\rm t}$ and a set of replacement candidates $\mathcal{M}$;
Output: a replacement scheduler;

```
1  S ← ∅;
2  let S_q be a scheduler queue;
3  S_q.push(∅);
4  while S_q is not empty do
5      s ← S_q.pop();
6      e ← next_conflict(Γ_t, M, s);
7      if e = −1 then
8          S ← S ∪ {s};
9          continue;
10     V ← compute the conflict set on index e;
11     for v ∈ V do
12         S_q.push(s ∪ {v});
13 return compute_depth(S);
```

The time complexity depends on the conflict policy. In the worst case, the precise policy is used and the worst time complexity is $O(|V|^m)$, where m is the

number of conflicts and $|V|$ is the size of the largest conflict set. When dealing with large-scale circuits, the precise policy is not scalable. Therefore we do not plan to demonstrate the precise policy in our experiments. The time complexity of both the greedy and stochastic policies are $O(m|V|)$.

Example 6. Continuing to consider Example 5, we show how to generate schedulers. Starting from the index $e = 0$, we search for the next conflicting index e in $\mathcal{M}$. When $e = 18$, the conflict set is $\{([10, 18], r_0, 18), ([18, 26, 31], r_3, 18)\}$. If the precise policy is used, we append the scheduler s with the two candidates and put it into the queue $\mathcal{S}_\mathrm{q}$. Then, the generated scheduler set $\mathcal{S} = \{s', s''\}$ are given as follows:

$$\begin{aligned} s' = \{&([0, 22], r_0), ([2, 13, 17], r_3), ([3, 9], r_0), \\ &([5, 12], r_1), ([10, 18], r_0, 18), ([15, 23], r_0), \\ &([16, 24], r_0), ([19, 25], r_1)\}, \\ s'' = \{&([0, 22], r_0), ([2, 13, 17], r_3), ([3, 9], r_0), \\ &([5, 12], r_1), ([18, 26, 31], r_3, 18), ([15, 23], r_0), \\ &([16, 24], r_0), ([19, 25], r_1)\}. \end{aligned}$$

If we use the greedy policy, the replacement candidate $([10, 18], r_0, 18)$ is selected, and the generated scheduler is s'. If we use the stochastic policy, one of them is selected and the finally generated scheduler is either s' or s''.

For the scheduler provided by Algorithm 3, we propose a replacement algorithm as shown in Algorithm 4, which reversely traverses each element of the scheduler and substitutes it according to the patterns. The input includes a target circuit $\mathcal{C}_\mathrm{t}$ and a scheduler s. The output is a substitution circuit. Line 2 obtains the mapping relationship Q_m of qubits between the subsequence of the target circuit and the pattern circuit, according to the qubit mapping function qubits_mapping(s_i). Line 3 updates the gates on the target circuit with the substitution circuit one by one. If the substitution gate sequence is longer than the pattern gate sequence, the redundant gates are inserted after the index where the pattern circuit appears in the target circuit. Otherwise, the redundant locations are removed from the target circuit. The time complexity is $O(m|s|)$, where m is the maximum length of the pattern circuit, and $|s|$ is the length of the scheduler s.

Algorithm 4: substitute($\mathcal{C}_\mathrm{t}$, s)

Input: a quantum circuit $\mathcal{C}_\mathrm{t}$, and a substitution scheduler s;
Output: the substituted circuit $\mathcal{C}_\mathrm{t}$;

```
1 for i ← |s| − 1 to 0 do
2     Q_m ← qubits_mapping(s_i);
3     C_t ←update(s_i, Q_m, C_t);
4 return C_t;
```

Example 7. Continuing to Example 6, we consider the replacement candidate $([2, 13, 17], r_3)$ as an example. We get the qubit mapping set $Q_\mathrm{m} = \{f(q_0) = q_3, f(q_1) = q_8\}$. The third gate of the target circuit $\mathcal{C}_\mathrm{t}$ is updated by the gate ("c", q_3q_8). The length of the pattern circuit is greater than that of the substitution circuit. Thus, the 13-th and 17-th gates are removed from the target circuit $\mathcal{C}_\mathrm{t}$.

5.3 Quantum Circuit Optimization

We develop a rule library for basic optimizations. The library is mainly used for basic reduction and quantum gate exchange[17, 33]. Note that almost all the gates implemented by quantum hardware devices are 1-qubit and 2-qubit gates, thus our rule library mainly concerns 1-qubit gates and 2-qubit gates. The maximum input scale involved in the rule set is up to 3-qubit gates. The gate specification involves some cancellation rules for 1-qubit gates and 2-qubit gates, as shown in Fig.5. The commutation rules shown in Fig.6 include transformation rules[33].

Sometimes, after a step of circuit rewriting the target circuit still matches some rules. We repeat several rounds of optimization and circuit rewriting until no pattern circuits can be matched or the specified repetition bound is reached (five by default in practice). It has been found experimentally that beyond five optimizations, the optimization opportunity decreases quickly. Hence, as a trade-off between optimization and running time, we choose to repeat the procedure five times.

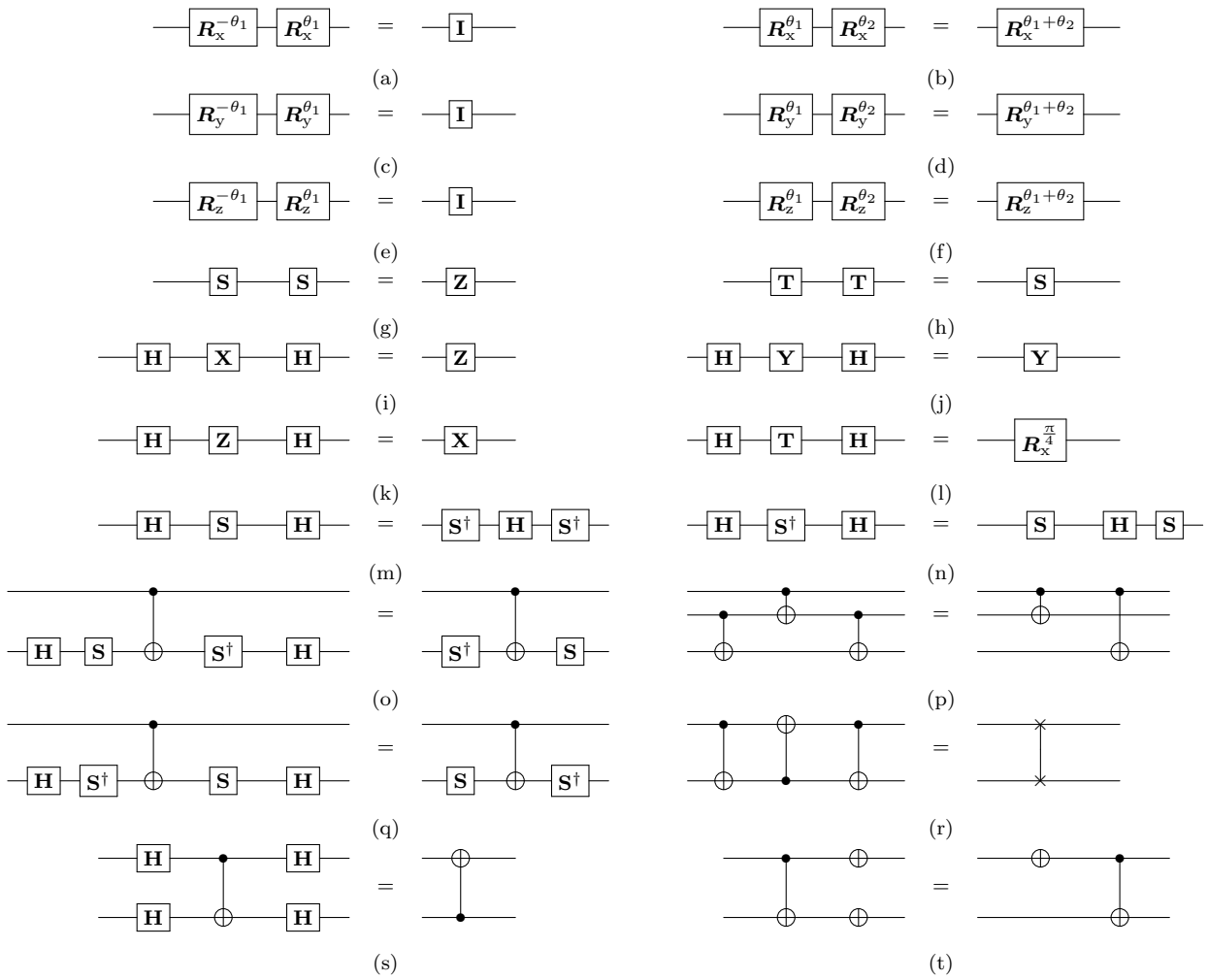


Fig.5. Gate cancellation rules. (a)–(n) 1-qubit gate rules. (o)–(t) 2-qubit gate rules.

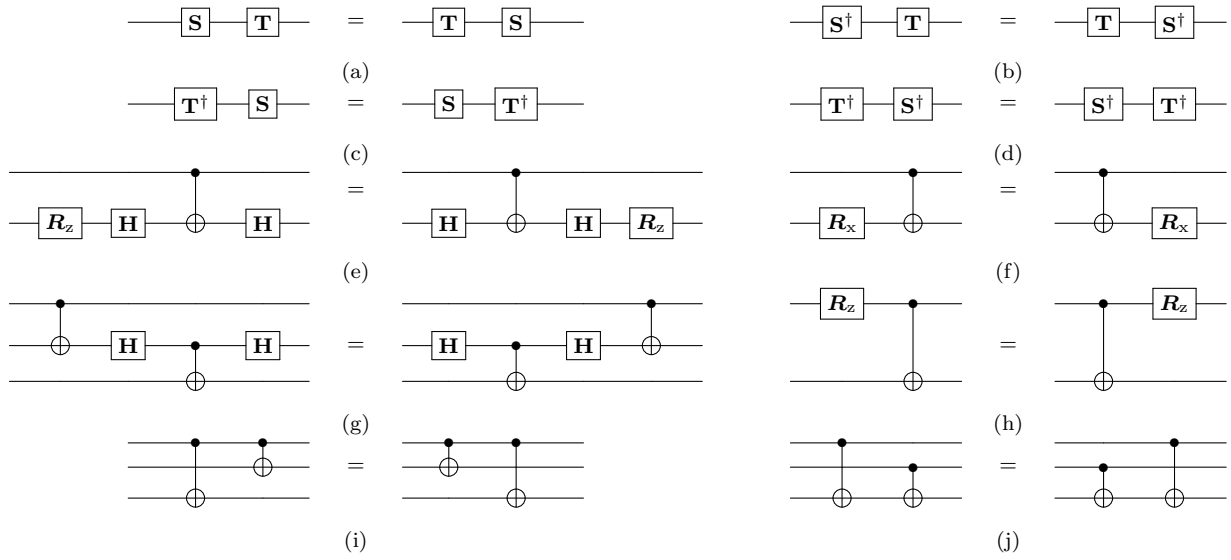


Fig.6. Commutation gate rules. (a)–(d) 1-qubit gate rules. (e)–(j) 2-qubit gate rules.

6 Case Studies

In this section, we consider two examples: the first rewrites a circuit with three **CCZ** gates to a circuit using the gate set G_{Sur}; the second optimizes a circuit with a stochastic policy.

6.1 Rewriting a Circuit for Surface-17

We demonstrate QRewriting using the gate decomposition rules in Fig.8 to rewrite the quantum circuit Toff-NC_3[16] shown in Fig.7 to the gate set G_{Sur}[10]. The **CCZ** gate decomposition rule[34] $r = (\mathcal{C}_{\text{p}}, \mathcal{C}_{\text{s}})$ is a pair, where

$$\begin{aligned}\mathcal{C}_{\text{p}} &= (\text{“E”}, q_0q_1q_2), \text{ and}\\ \mathcal{C}_{\text{s}} &= (\text{“t”}, q_0)(\text{“t”}, q_1)(\text{“c”}, q_2q_0)(\text{“c”}, q_1q_2)(\text{“T”}, q_0)\\ &\quad (\text{“T”}, q_2)(\text{“c”}, q_1q_0)(\text{“c”}, q_1q_2)(\text{“t”}, q_0)(\text{“c”}, q_2q_0)\\ &\quad (\text{“T”}, q_0)(\text{“t”}, q_2)(\text{“c”}, q_1q_0).\end{aligned}$$

The gate sequences of the target circuit and the decomposition pattern circuit are “hEhhEhhEh” and “E”, respectively. The subsequence set of $\mathcal{C}_{\text{t}}$ that can match “E” is $\{[1], [4], [7]\}$. Finally, the resulting circuit is shown in Fig.9.

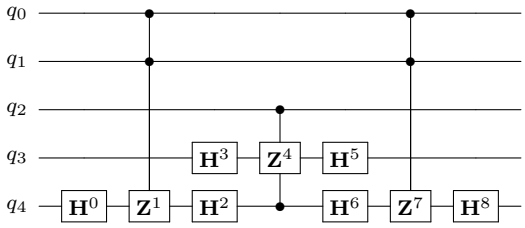


Fig.7. The quantum circuit Toff-NC_3. The markers 0–8 on the gate symbols represent the order in the gate sequence.

6.2 Optimization with a Stochastic Policy

Next, we show an example of circuit optimization using QSRewriting with a stochastic policy. The target circuit and a set of rules $\mathcal{R} = \{r_0, r_1, r_2, r_3\}$ are shown in Fig.4(a) and Fig.1, respectively. The gate sequences of the target circuit and pattern circuits are “xxxxccccxxxxxcccxxxxcccxxxxxxccc”, “xx”, “cc”, “ccc”, and “xcx”, respectively. The subsequence sets of the gate sequence of the pattern circuits are as follows:

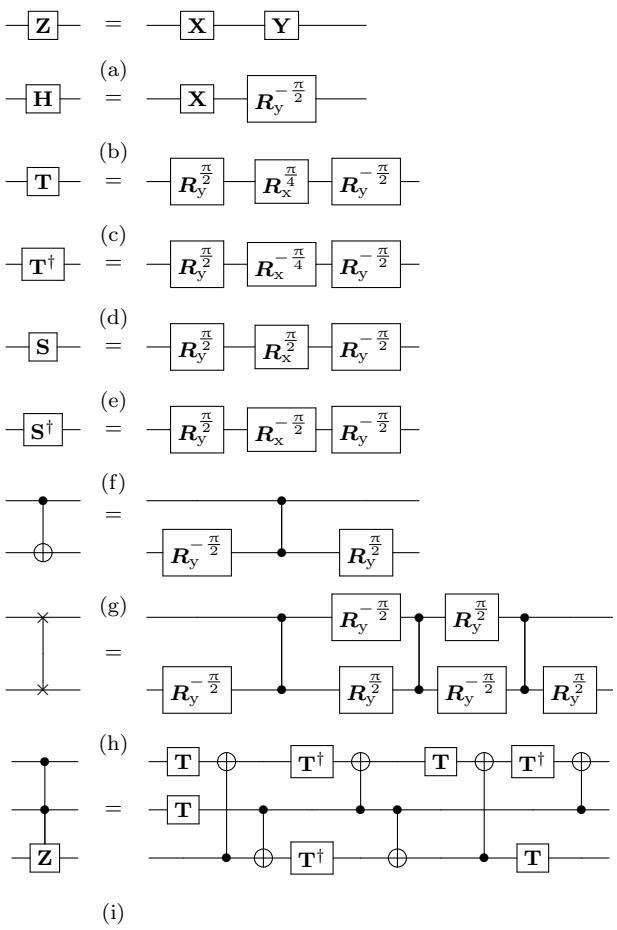


Fig.8. Gate decomposition into primitives supported in the superconducting Surface-17 processor. (a)–(f) 1-qubit gate rules. (g)–(h) 2-qubit gate rules. (i) 3-qubit gate rule.

- “xx”: $\{[3, 9], [10, 18], [0, 22], [15, 23], [16, 24]\}$,
- “cc”: $\{[5, 12], [19, 25]\}$,
- “ccc”: $\emptyset$,
- “xcx”: $\{[2, 13, 17], [18, 26, 31]\}$.

In Fig.4(a), we have marked the order of the elements in subsequence sets on the circuit symbols. The replacement candidates $([10, 18], r_0, 18)$ and $([18, 26, 31], r_3, 18)$ have a conflict at the index 18 of the target circuit. With the stochastic policy, either of the candidates can be chosen. Suppose the former is taken, then the generated replacement scheduler is given as follows:

$$\begin{aligned}s = \{&([0, 22], r_0), ([2, 13, 17], r_3), ([3, 9], r_0),\\ &([5, 12], r_1), ([10, 18], r_0, 18), ([15, 23], r_0),\\ &([16, 24], r_0), ([19, 25], r_1)\}.\end{aligned}$$

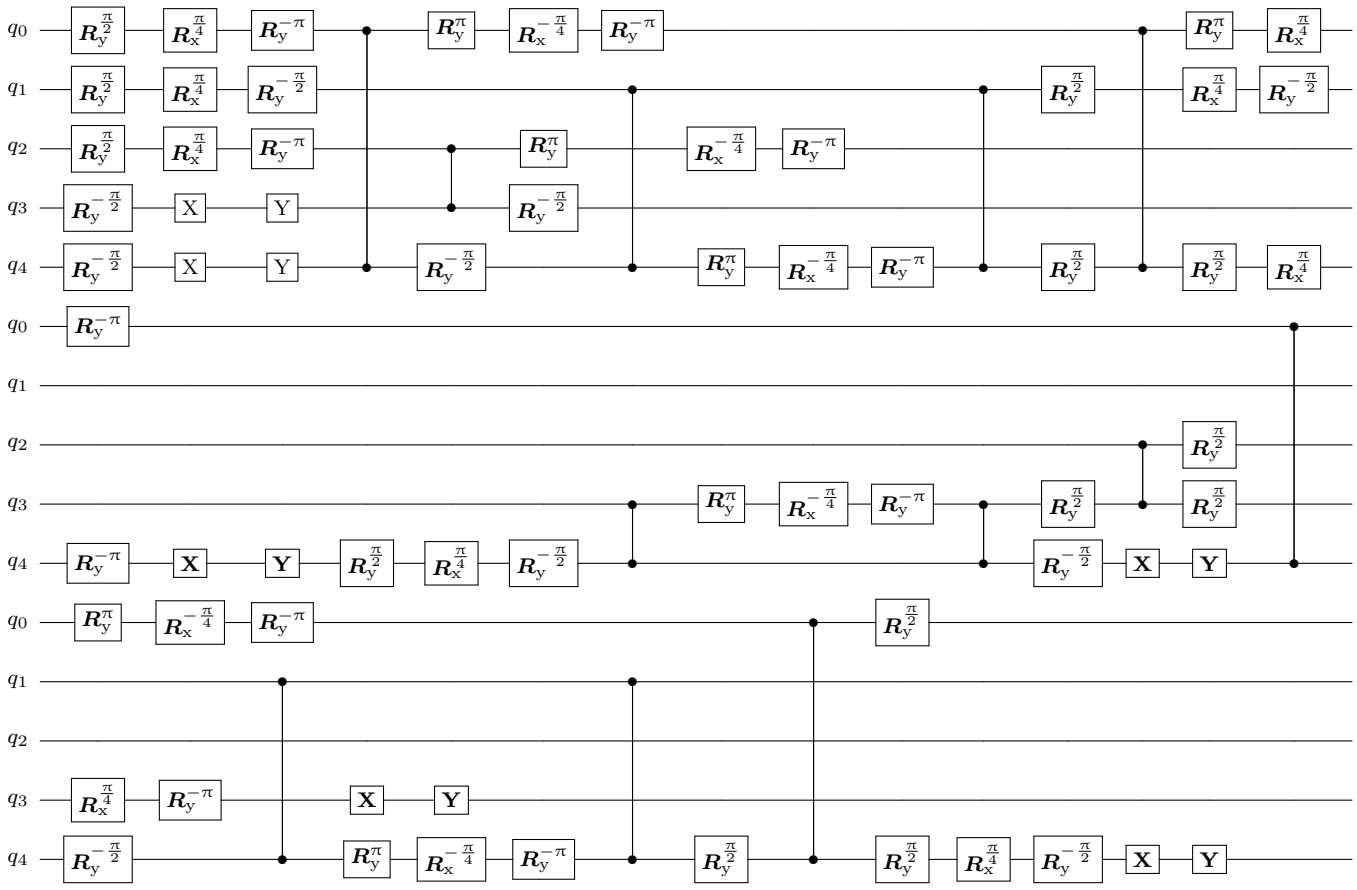


Fig.9. The quantum circuit Toff-NC$_3$ rewritten to the Surface-17 quantum processor.

Finally, we obtain the resulting circuit shown in Fig.4(b), which reduces the gate count and the depth by 49% and 20%, respectively.

7 Experiments

We compare QRewriting with a state-of-the-art algorithm for quantum circuit optimization framework based on pattern matching, namely PaF[29]. Note that PaF is not freely available, thus we implement that algorithm in Python. The implementation of QRewriting in Python is available online④. All the experiments are conducted on a Ubuntu machine with a 2.2GHz CPU and 64G memory. For the stochastic policy, we execute QSRewriting five times and take the average result; for other policies are deterministic, thus we execute them only once.

We compare QSRewriting, QGRewriting, and PaF, using the BIGD[16] benchmarks and the rules shown in Fig.1, with the results shown in Fig.10. The depth and the gate count of the generated quantum circuits are used as evaluation metrics. The BIGD benchmarks are characterized by the parameter (v_1, v_2), which is called the gate density vector[16]. The two components stand for the densities of 1-qubit and 2-qubit gates of a circuit, respectively. Suppose a quantum circuit has n qubits, N_{1q} (resp. N_{2q}) is the number of 1-qubit (resp. 2-qubit) gates, and the longest dependency chain is l, then $v_1 = N_{1q}/(n \times l)$ and $v_2 = 2 \times N_{2q}/(n \times l)$.

The BIGD benchmarks include 360 circuits with a total of 129600 gates. After a PaF optimization, the gate count and the depth decrease by 66236 and 3999 within 7125 seconds, respectively. QGRewriting (resp. QSRewriting) takes 1387 (resp. 1509) seconds

④https://github.com/Holly-Jiang/QRewriting.git, Jun. 2022.

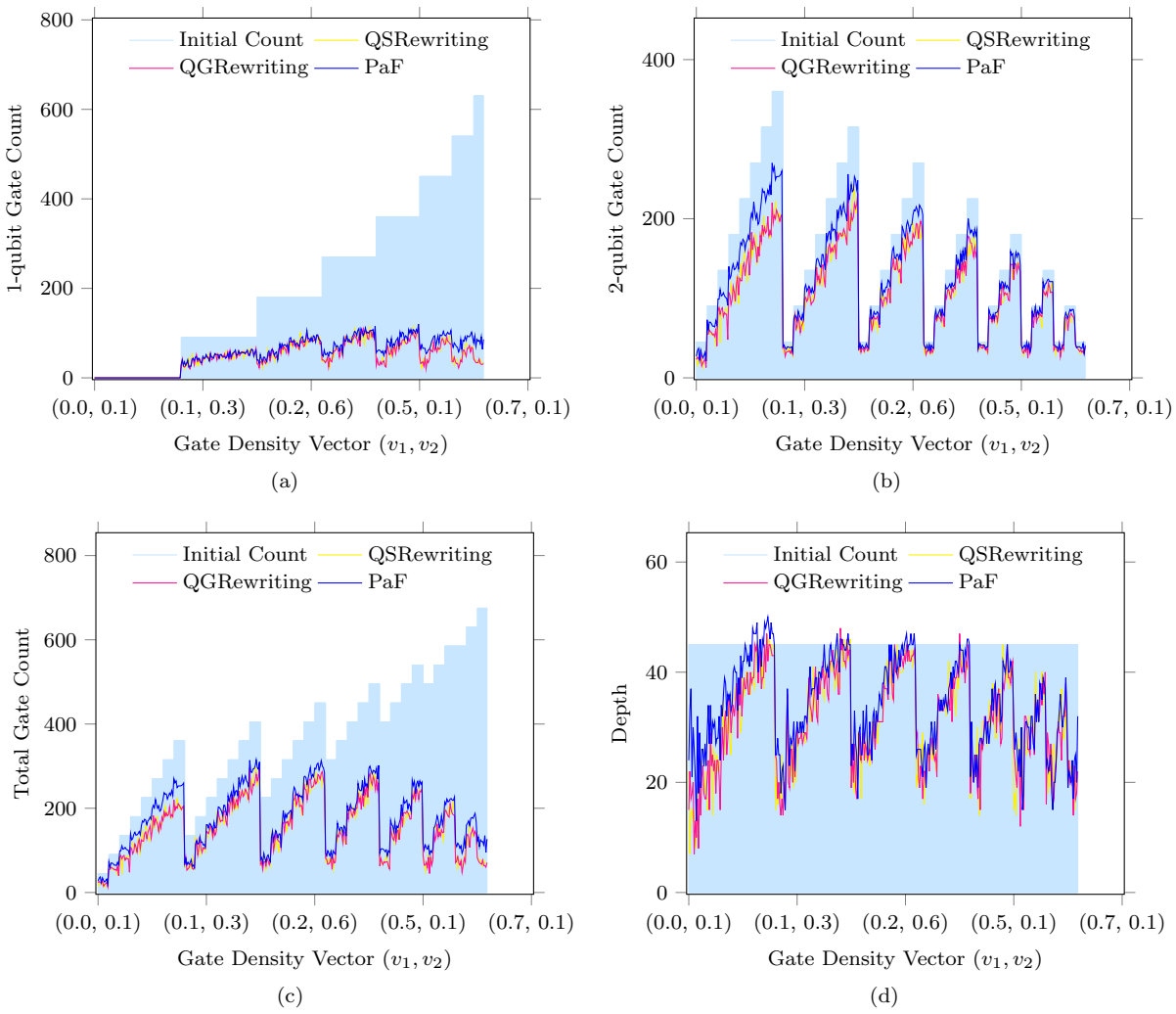


Fig.10. Comparison of the quantum circuits generated by QSRewriting, QGRewriting, and PaF optimized on the BIGD[16] benchmarks. (a) 1-qubit gate count. (b) 2-qubit gate count. (c) Total gate count. (d) Depth.

to rewrite these benchmark circuits, and the generated circuits further reduce the 1-qubit gate count, 2-qubit gate count, total gate count, and depth by an average of 5.2% (resp. 4.5%), 57.0% (resp. 55.9%), 17.4% (resp. 16.7%), and 26.5% (resp. 24.8%) compared with PaF. The main evaluation results are shown in Fig.10, which compares the performance of QSRewriting, QGRewriting, and PaF in terms of 1-qubit gate count, 2-qubit gate count, total gate count, and depth of the generated circuits. The light blue bars represent the gate count (depth) of the benchmarks. The blue, red, and yellow lines represent PaF, QGRewriting, and QSRewriting, respectively. We can see that the red and yellow lines are mostly lower than the blue, and the yellow is mostly obscured by the red, but we can still see that it is lower than the red in some places. In Fig.10(d), we can see that in a few cases, the depth of those quantum circuits might increase after optimizing. The reason is that the gates of a rear layer may be moved to the front layer, causing the original gates of the front layer to conflict with them.

In Fig.10, QSRewriting is lower than QGRewriting on some BIGD benchmarks on which we compare the gate count and depth increments of the circuits generated by QSRewriting and QGRewriting, as shown in Fig.11(a) and Fig.11(b), respectively. In Fig.11, the light blue bars represent the total gate count (depth)

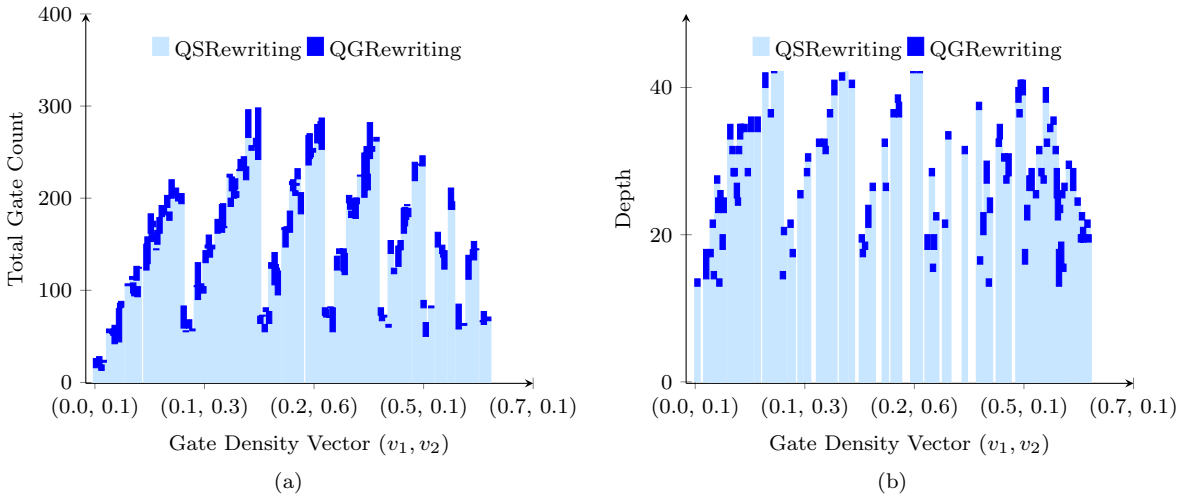


Fig.11. Comparison of the increments of the generated circuits on which QSRewriting outperforms QGRewriting both optimized on the BIGD[16] benchmarks. (a) Total gate count. (b) Depth.

of the circuits generated by QSRewriting and the blue parts are the increments in gate count (depth) that QGRewriting has over QSRewriting. The greedy policy is deterministic and the stochastic policy shows that better strategies exist. In Fig.12, we show a comparison of the time cost of the three methods optimized on the BIGD benchmarks, with PaF in blue, QGRewriting in red, and QSRewriting in yellow. It is clear that QGRewriting and QSRewriting use less time, as both the red and yellow lines are lower than the blue line in Fig.12. On average, both QGRewriting and QSRewriting optimized on the BIGD benchmarks are about 5 times faster than PaF.

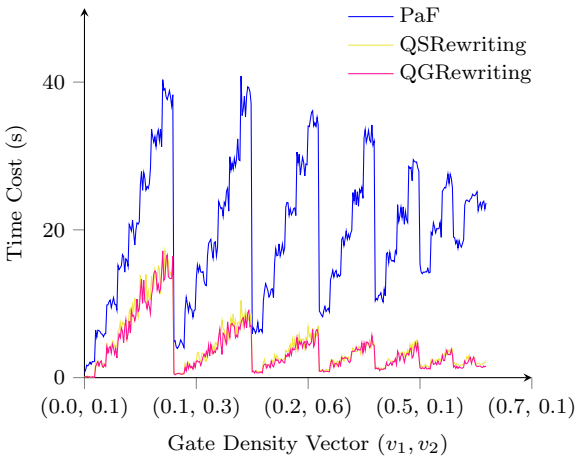


Fig.12. Comparison of the time cost for QSRewriting, QGRewriting, and PaF optimized on the BIGD[16] benchmarks.

Now we consider rewriting a set of benchmark circuits[17] consisting of arithmetic circuits and implementations of multi-controlled Toffoli gates to the Surface-17 processor, as shown in Tables 3 and 4. The set of benchmark circuits uses the commonly used gate set G_{Com} (see Table 1) with a total of 33 circuits and 201554 gates. The gate set G_{Sur} supported by Surface-17 processor limits 1-qubit gates to $\boldsymbol{R}_{\text{x}}$ and $\boldsymbol{R}_{\text{y}}$ rotations and 2-qubit **CZ** gate, and more specifically $\pm\frac{\pi}{4}$, $\pm\frac{\pi}{2}$, and $\pm\pi$ degrees will be used in the decomposition, as shown in Fig.8. Thus, the rewriting of the benchmark circuits is from the gate set G_{Com} to the gate set G_{Sur}, and the gate count (resp. depth) and time cost for each phase are shown in Table 3 (resp. Table 4) in detail. Note that the rewriting of the benchmarks simply chooses the greedy policy since no replacement conflicts arise. We calculate the reduction rate of gate count (resp. depth) by optimizing each rewritten benchmark, as shown in the last column of Table 3 (resp. Table 4). The optimization of the rewritten circuits leads to a reduction of up to 52% in gate count and 49% in depth. However, there is a price to pay. For example, for the benchmark circuit "GF(2^{163})-Mult" in the last row of Table 3 with millions of gates, the rewriting without optimization takes about 15 minutes (see the last row of column t_1), while the rewriting with optimization may take about two hours (see the last row of column

Table 3. Comparison of the Gate Count of the Circuits Generated by QRewriting from the Benchmark Circuits[17]

Benchmark	n	N	N_0	t_0	N_1	t_1	N_2	t_2	$\Delta(\%)$
Toff-NC_3	5	9	45	0.00	135	0.03	80	0.50	40.74
Toff-$Barenco_3$	5	10	58	0.00	174	0.07	101	0.52	41.95
Mod 5_4	5	15	63	0.00	187	0.09	89	0.57	**52.41**
Toff-NC_4	7	15	75	0.00	225	0.10	134	0.89	40.44
Toff-$Barenco_4$	7	18	114	0.00	342	0.19	198	1.12	42.11
Toff-NC_5	9	21	105	0.00	315	0.18	188	1.34	40.32
Toff-$Barenco_5$	9	26	170	0.01	510	0.32	296	1.70	41.96
VBE-$Adder_3$	10	30	150	0.00	450	0.34	266	1.51	40.89
GF(2^4)-Mult	12	33	225	0.01	675	0.58	388	2.68	42.52
Mod-$Mult_{55}$	9	35	119	0.00	341	0.20	211	1.04	38.12
GF(2^5)-Mult	15	47	347	0.01	1041	0.80	601	3.74	42.27
CSLA-MUX_3	15	50	170	0.01	510	0.44	315	2.23	38.24
Toff-NC_{10}	19	51	255	0.01	765	0.49	458	3.31	40.13
GF(2^6)-Mult	18	63	495	0.02	1485	1.19	854	5.36	42.49
Toff-$Barenco_{10}$	19	66	450	0.01	1350	0.95	786	4.76	41.78
RC-$Adder_6$	14	68	200	0.01	584	0.93	361	2.66	38.18
Mod-Red_{21}	11	74	278	0.01	786	0.93	463	3.39	41.09
GF(2^7)-Mult	21	81	669	0.02	2007	1.61	1153	7.37	42.55
CSUM-MUX_9	30	84	420	0.01	1204	1.57	721	3.98	40.12
QCLA-Com_7	24	95	443	0.01	1299	1.02	778	5.98	40.11
QCLA-$Adder_{10}$	36	113	521	0.01	1563	1.31	957	7.29	38.77
GF(2^8)-Mult	24	115	883	0.02	2649	2.35	1516	12.95	42.77
GF(2^9)-Mult	27	123	1095	0.03	3285	2.72	1885	12.09	42.62
GF(2^{10})-Mult	30	147	1347	0.03	4041	3.36	2316	14.90	42.69
QCLA-Mod_7	26	176	884	0.02	2638	2.06	1570	12.24	40.49
$Adder_8$	24	216	900	0.02	2676	6.79	1623	13.00	39.35
GF(2^{16})-Mult	48	363	3435	0.13	10305	9.44	5865	38.96	43.09
Mod-$Adder_{1024}$	28	865	4285	0.09	12855	18.19	7403	57.77	42.41
GF(2^{32})-Mult	96	1305	13593	0.30	40779	38.71	23069	157.25	43.43
GF(2^{64})-Mult	192	4539	53691	1.17	161073	146.63	91065	635.20	43.46
GF(2^{128})-Mult	384	17275	213883	5.59	641649	584.97	362429	3656.48	43.52
GF(2^{131})-Mult	393	18333	224265	5.90	672795	616.90	379766	3927.99	43.55
GF(2^{163})-Mult	489	27705	346533	9.67	1039599	945.51	587034	7452.82	43.53

Note: n: the number of qubits. N: the gate count of the circuit. N_0: the gate count of the circuit after decomposition without optimization on gate set G_{Com}. N_1: the gate count of the circuit after rewriting without optimization on gate set G_{Sur}. N_2: the gate count of the circuit after rewriting with optimization on gate set G_{Sur}. $t_i\,(i=0,1,2)$: running time in seconds. Δ: $(N_1-N_2)/N_1\times 100\%$.

t_2), depending on the size of the rule library and the structure of the quantum circuit.

8 Conclusion

We introduced a new representation of quantum circuits, which reduces the pattern matching of quantum circuits to the problem of finding distinct subsequences. We presented an algorithm based on dynamic programming to match the pattern circuits in the target circuit. To resolve replacement conflicts, we proposed three policies for generating a replacement scheduler and a polynomial-time replacement algorithm. We developed a rule library for basic optimizations and applied it to rewrite the benchmarks consisting of arithmetic circuits and implementations of multi-controlled Toffoli gates to the Surface-17 processor. Compared with the existing method PaF optimized on the BIGD benchmarks, QRewriting reduces the depth (resp. gate count) by 26.5% (resp. 17.4%), which demonstrates the effectiveness of the proposed method.

Table 4. Comparison of the Depth of the Circuits Generated by QRewriting from the Benchmark Circuits[17]

Benchmark	n	d	d_0	t_0	d_1	t_1	d_2	t_2	$\Delta(\%)$
Toff-NC_3	5	7	23	0.00	64	0.03	42	0.50	34.38
Toff-$Barenco_3$	5	9	31	0.00	86	0.07	52	0.52	39.53
Mod 5_4	5	15	36	0.00	97	0.09	49	0.57	**49.48**
Toff-NC_4	7	11	38	0.00	104	0.10	67	0.89	35.58
Toff-$Barenco_4$	7	17	61	0.00	166	0.19	102	1.12	38.55
Toff-NC_5	9	15	53	0.00	144	0.18	92	1.34	36.11
Toff-$Barenco_5$	9	25	91	0.01	246	0.32	152	1.70	38.21
VBE-$Adder_3$	10	20	70	0.00	194	0.34	113	1.51	41.75
GF(2^4)-Mult	12	17	85	0.01	236	0.58	145	2.68	38.56
Mod-$Mult_{55}$	9	14	43	0.00	118	0.20	80	1.04	32.20
GF(2^5)-Mult	15	20	111	0.01	310	0.80	187	3.74	39.68
CSLA-MUX_3	15	17	59	0.01	166	0.44	107	2.23	35.54
Toff-NC_{10}	19	35	128	0.01	344	0.49	217	3.31	36.92
GF(2^6)-Mult	18	25	139	0.02	384	1.19	235	5.36	38.80
Toff-$Barenco_{10}$	19	65	241	0.01	646	0.95	402	4.76	37.77
RC-$Adder_6$	14	28	93	0.01	261	0.93	166	2.66	36.40
Mod-Red_{21}	11	43	141	0.01	383	0.93	238	3.39	37.86
GF(2^7)-Mult	21	29	166	0.02	458	1.61	280	7.37	38.86
CSUM-MUX_9	30	15	53	0.01	147	1.57	96	3.98	34.69
QCLA-Com_7	24	15	70	0.01	192	1.02	115	5.98	40.10
QCLA-$Adder_{10}$	36	15	64	0.01	182	1.31	111	7.29	39.01
GF(2^8)-Mult	24	39	199	0.02	544	2.35	335	12.95	38.42
GF(2^9)-Mult	27	36	219	0.03	606	2.72	367	12.09	39.44
GF(2^{10})-Mult	30	40	246	0.03	680	3.36	412	14.90	39.41
QCLA-Mod_7	26	39	172	0.02	487	2.06	284	12.24	41.68
$Adder_8$	24	55	191	0.02	527	6.79	315	13.00	40.23
GF(2^{16})-Mult	48	71	415	0.13	1136	9.44	699	38.96	38.47
Mod-$Adder_{1024}$	28	521	2218	0.09	6397	18.19	3775	57.77	40.99
GF(2^{32})-Mult	96	137	849	0.30	2324	38.71	1447	157.25	37.74
GF(2^{64})-Mult	192	263	1711	1.17	4688	146.63	2856	635.20	39.08
GF(2^{128})-Mult	384	517	3437	5.59	9420	584.97	5750	3656.48	38.96
GF(2^{131})-Mult	393	537	3526	5.90	9658	616.90	5902	3927.99	38.89
GF(2^{163})-Mult	489	665	4390	9.67	12026	945.51	7310	7452.82	39.22

Note: n: the number of qubits. d: the depth of the circuit. d_0: the depth of the circuit after decomposition without optimization on gate set G_{Com}. d_1: the depth of the circuit after rewriting without optimization on gate set G_{Sur}. d_2: the depth of the circuit after rewriting with optimization on gate set G_{Sur}. t_i $(i = 0, 1, 2)$: running time in seconds. Δ: $(d_1 - d_2)/d_1 \times 100\%$.

References

[1] Shor P W. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proc. the 35th Annual Symposium on Foundations of Computer Science*, 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700.

[2] Grover L K. A fast quantum mechanical algorithm for database search. In *Proc. the 28th Annual ACM Symposium on the Theory of Computing*, 1996, pp. 212–219. DOI: 10.1145/237814.237866.

[3] Harrow A W, Hassidim A, Lloyd S. Quantum algorithm for solving linear systems of equations. *Physical Review Letters*, 2009, 103(15):Article No. 150502. DOI: 10.1103/PhysRevLett.103.150502.

[4] Arute F, Arya K, al. Quantum supremacy using a programmable superconducting processor. *Nature*, 2019, 574:505–510. DOI: 10.1038/s41586-019-1666-5.

[5] Madsen L S, Laudenbach F, Askarani M F, Rortais F, Vincent T, Bulmer J F F, Miatto F M, Neuhaus L, Helt L G, Collins M J, Lita A E, Gerrits T, Nam S W, Vaidya V D, Menotti M, Dhand I, Vernon Z, Quesada N,

Lavoie J. Quantum computational advantage with a programmable photonic processor. *Nature*, 2022, 606:75–81. DOI: 10.1038/s41586-022-04725-x.

[6] Preskill J. Quantum Computing in the NISQ era and beyond. *Quantum*, 2018, 2:Article No. 79. DOI: 10.22331/q-2018-08-06-79.

[7] Kjaergaard M, Schwartz M E, Braumüller J, Krantz P, Wang J I J, Gustavsson S, Oliver W D. Superconducting qubits: Current state of play. *Annual Review of Condensed Matter Physics*, 2020, 11(1):369–395. DOI: 10.1146/annurev-conmatphys-031119-050605.

[8] Nielsen M A, Chuang I L. *Quantum Computation and Quantum Information (10th Anniversary edition)*. Cambridge University Press, 2016.

[9] Murali P, Linke N M, Martonosi M, Abhari A J, Nguyen N H, Alderete C H. Full-stack, real-system quantum computer studies: Architectural comparisons and design insights. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, 2019, pp. 527–540. DOI: 10.1145/3307650.3322273.

[10] Lao L, Someren H, Ashraf I, Almudéver C G. Timing and resource-aware mapping of quantum circuits to superconducting processors. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 2022, 41(2):359–371. DOI: 10.1109/TCAD.2021.3057583.

[11] Siraichi M Y, Santos V F, Collange S, Pereira F M Q. Qubit allocation. In *Proc. the 2018 International Symposium on Code Generation and Optimization*, 2018, pp. 113–125. DOI: 10.1145/3168822.

[12] Li G, Ding Y, Xie Y. Tackling the qubit mapping problem for NISQ-era quantum devices. In *Proc. the 24th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 1001–1014. DOI: 10.1145/3297858.3304023.

[13] Liu L, Dou X. Qucloud: A new qubit mapping mechanism for multi-programming quantum computing in cloud environment. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, pp. 167–178. DOI: 10.1109/HPCA51647.2021.00024.

[14] Liu L, Dou X. Qucloud+: A holistic qubit mapping scheme for single/multi-programming on 2d/3d nisq quantum computers. *ACM Trans. Archit. Code Optim.*, 2023. DOI: 10.1145/3631525.

[15] Childs A M, Schoute E, Unsal C M. Circuit Transformations for Quantum Architectures. In *the 14th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2019)*, volume 135, 2019, pp. 3:1–3:24. DOI: 10.4230/LIPIcs.TQC.2019.3.

[16] Tan B, Cong J. Optimality study of existing quantum computing layout synthesis tools. *IEEE Transactions on Computers*, 2020, 70(9):1363–1373. DOI: 10.1109/TC.2020.3009140.

[17] Nam Y S, Ross N J, Su Y, Childs A M, Maslov D. Automated optimization of large quantum circuits with continuous parameters. *npj Quantum Information*, 2018, 4:Article No. 23. DOI: 10.1038/s41534-018-0072-4.

[18] Sivarajah S, Dilkes S, Cowtan A, Simmons W, Edgington A, Duncan R. t|ket⟩: a retargetable compiler for NISQ devices. *Quantum Science and Technology*, 2020, 6(1):Article No. 014003. DOI: 10.1088/2058-9565/ab8e92.

[19] Jia Z, Padon O, Thomas J, Warszawski T, Zaharia M, Aiken A. Taso: Optimizing deep learning computation with automatic generation of graph substitutions. In *Proc. the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 47–62. DOI: 10.1145/3341301.3359630.

[20] Kissinger A, Wetering J. PyZX: Large scale automated diagrammatic reasoning. *Electronic Proceedings in Theoretical Computer Science*, 2020, 318:229–241. DOI: 10.4204/EPTCS.318.14.

[21] Pointing, Jessica and Padon, Oded and Jia, Zhihao and Ma, Henry and Hirth, Auguste and Palsberg, Jens and Aiken, Alex. Quanto: Optimizing Quantum Circuits with Automatic Generation of Circuit Identities. arXiv: 1908.08963, 2021. https://arxiv.org/abs/2111.11387, Jun. 2022.

[22] Xu M, Li Z, Padon O, Lin S, Pointing J, Hirth A, Ma H, Palsberg J, Aiken A, Acar U A, Jia Z. Quartz: Superoptimization of quantum circuits. In *Proc. the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, 2022, pp. 625–640. DOI: 10.1145/3519939.3523433.

[23] McKeeman W M. Peephole optimization. *Commun. ACM*, 1965, 8(7):443–444. DOI: 10.1145/364995.365000.

[24] Abdessaied N, Soeken M, Wille R, Drechsler R. Exact template matching using boolean satisfiability. In *2013 IEEE 43rd International Symposium on Multiple-Valued Logic*, 2013, pp. 328–333. DOI: 10.1109/ISMVL.2013.26.

[25] Iten R, Moyard R, Metger T, Sutter D, Woerner S. Exact and practical pattern matching for quantum circuit optimization. *ACM Transactions on Quantum Computing*, 2022, 3(1):Article No. 4. DOI: 10.1145/3498325.

[26] Rahman M M, Dueck G W. Optimal quantum circuits of three qubits. In *2012 IEEE 42nd International Symposium on Multiple-Valued Logic*, 2012, pp. 161–166. DOI: 10.1109/ISMVL.2012.43.

[27] Prasad A K, Shende V V, Markov I L, Hayes J P, Patel K N. Data structures and algorithms for simplifying reversible circuits. *ACM J. Emerg. Technol. Comput. Syst.*, 2006, 2(4):277–293. DOI: 10.1145/1216396.1216399.

[28] Soeken M, Dueck G W, Rahman M M, Miller D M. An extension of transformation-based reversible and quantum circuit synthesis. In *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2016, pp. 2290–2293. DOI: 10.1109/ISCAS.2016.7539041.

[29] Chen M, Zhang Y, Li Y. A quantum circuit optimization framework based on pattern matching. *SPIN*, 2021, 11(03):Article No. 2140008. DOI: 10.1142/S2010324721400087.

[30] Wagner R A, Fischer M J. The string-to-string correction problem. *J. ACM*, 1974, 21(1):168–173. DOI: 10.1145/321796.321811.

[31] Bellman R. Dynamic programming. *Science*, 1966, 153(3731):34–37. DOI: 10.1126/science.153.3731.34.

[32] Zhang Y, Deng H, Li Q, Song H, Nie L. Optimizing quantum programs against decoherence: Delaying qubits into quantum superposition. In *2019 International Symposium on Theoretical Aspects of Software Engineering*, 2019, pp. 184–191. DOI: 10.1109/TASE.2019.000-2.

[33] Iwama K, Kambayashi Y, Yamashita S. Transformation rules for designing cnot-based quantum circuits. In *Proc. the 39th Annual Design Automation Conference*, 2002, pp. 419–424. DOI: 10.1145/513918.514026.

[34] Amy M, Azimzadeh P, Mosca M. On the controlled-NOT complexity of controlled-NOT–phase circuits. *Quantum Science and Technology*, 2018, 4(1):Article No. 015002. DOI: 10.1088/2058-9565/aad8ca.

Hui Jiang received her B.Eng. degree in computer science and technology from Sichuan Agriculture University, Ya'an, in 2019. She is currently a Ph.D. candidate at Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai. Her research interests include quantum circuit compilation and optimization.

Dian-Kang Li received his bachelor of science degree in educational technology in 2020, and M.E. in software engineering in 2023, both from East China Normal University, Shanghai. He is currently an engineer at Meituan. His primary research interests lie in the fields of quantum machine learning and quantum deep learning.

Yu-Xin Deng received his B.Eng. degree in thermal energy engineering and M.Sc. degree in computer science from Shanghai Jiao Tong University, Shanghai, in 1999 and 2002, respectively, and Ph.D. degree in computer science from Ecole des Mines de Paris, Paris, in 2005. He is a professor at East China Normal University, Shanghai. His research interests include concurrency theory, especially about process calculi, formal semantics of programming languages, as well as quantum computing. He authored the book titled Semantics of Probabilistic Processes: An Operational Approach (Springer, 2015).

Ming Xu received his B.Eng degree in software engineering and Ph.D. degree in system sciences from East China Normal University (ECNU), Shanghai, in 2005 and 2010, respectively. He is currently an associate research professor at Shanghai Key Laboratory of Trustworthy Computing, ECNU. His research interests include computer algebra, program verification, and quantum computing.