A Binary Integer Programming-Based Method for Qubit Mapping in Sparse Architectures

Hui Jiang[1], Jianling Fu[2], Yuxin Deng[1*], Jun Wu[1]

[1*]Shanghai Key Lab of Trustworthy Computing, East China Normal University, Shanghai, 200063, China.
[2]Shanghai Institute for AI Education, East China Normal University, Shanghai, 200063,China.

*Corresponding author(s). E-mail(s): yxdeng@sei.ecnu.edu.cn;
Contributing authors: yhq_jh@126.com; scsse_fjl2015@126.com;
51215902063@stu.ecnu.edu.cn;

Abstract

It is a current trend of sparse architectures employed for superconducting quantum chips, which have the advantage of low coupling and crosstalk properties. Existing qubit mapping algorithms do not take the sparsity of quantum architectures into account. To this end, we propose a qubit mapping method based on binary integer programming, called QMBIP. First, we slice a given quantum circuit by taking into account the sparsity of target architectures. Then, the constraints and the objective function are formulated and rendered to the binary integer programming problem by matrix transformation. The behavior of a **SWAP** gate is characterized by an elementary row transformation on the mapping matrix between the physical and logical qubits. To reduce the search space, we introduce path variables and isomorphic pruning, as well as a look-ahead mechanism. Finally, we compare with typical qubit mapping algorithms such as SABRE and SATMAP on the sparse architectures *ibmq_sydney*, *ibmq_manhattan*, *ibmq_singapore*, and a dense architecture *ibmq_tokyo*. Experiments show that QMBIP effectively maintains the fidelity of the compiled quantum circuits. For example, on *ibmq_sydney*, the fidelity of the quantum circuits compiled by our approach outperforms SABRE and SATMAP by 53.9% and 46.8%, respectively.

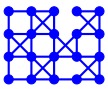
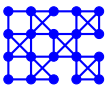
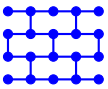
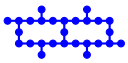
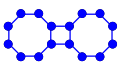
Keywords: qubit mapping, sparse architecture, binary integer programming

1 Introduction

Over the past four decades, quantum computing has been evolving at a rapid pace. In 1980, Paul Benioff proposed the quantum mechanical model of Turing machines [1]. During the last two decades, tech giants have developed prototypes of quantum computers aiming to solve specific tasks. For example, in late 2023, IBM released the first-ever 1,000-qubit superconducting quantum chip "Condor"[1].

There are several ways of building a quantum computer, e.g., ion-trap quantum computers [2], superconducting quantum circuits [3] and photonic quantum devices [4]. Superconducting quantum circuits are widely investigated because they enjoy good scalability, but in which a qubit is only connected to its adjacent qubits. Table 1 shows the evolution of the topology and the average qubit connectivity for superconducting quantum processors. As we can see, the trend is to lower the average connectivity of qubits. The current adoption of such a sparse architecture of quantum chips brings more scalability, less error rates, and more opportunities to explore error-correcting codes.

Table 1 The evolution of the topology for superconducting quantum systems (left to right)

processor	Penguin v1	Penguin v2	Penguin v4	Falcon r4[2]	Aspen-4[3]
architecture					
average qubit connectivity	3.9	3.7	2.3	2.1	2.25

The design of quantum algorithms usually leaves aside hardware constraints, but physical constraints must be considered when compiling these algorithms. It is well known that qubit mapping is an NP-complete problem [5]. Paler [6] has shown that initial mappings have an important impact on qubit mapping. SABRE [7] depends on a random initial mapping. SAHS [8] uses an annealing algorithm to find an initial mapping, but it is unstable. Siraichi *et al.* constructed a weighted dependence graph [5] according to the degree of qubits. Subgraph isomorphism performs well in the initial mapping, which considers the whole quantum circuit [9]. Much previous work has reduced qubit mapping problems to existing ones, such as AI planning [10, 11], integer programming [12] and boolean satisfiability problem (SAT) [13, 14], and then used corresponding tools to find an optimal solution for the problem in an acceptable amount of time consumption. Different from the above approaches, in this work we employ the method of encoding a path as a variable, instead of the usual approach of encoding a **SWAP** gate as a variable, which greatly prunes the search space. Exact algorithms are often sensitive to the size of a quantum circuit as well as the chip architecture. In order to handle larger circuits, heuristic algorithms for qubit mapping have appeared

[1] https://www.nature.com/articles/d41586-021-03476-5, accessed 15 March 2023.
[2] https://research.ibm.com/blog/heavy-hex-lattice/, accessed 17 December 2023.
[3] https://qcs.rigetti.com/qpus, accessed 17 December 2023.

in the literature, which selects a path towards satisfying the objective function. Some heuristic algorithms aim at inserting as few **SWAP** gates as possible [5, 7–9, 15–18], maximizing the fidelity of the compiled circuits [19] or minimizing the overall circuit latency [20, 21]. However, we observed that existing heuristic algorithms do not perform well on circuits of sparse architecture if we consider the fidelity of the compiled quantum circuits. Note that fidelity is the most accurate to evaluate the quality of quantum circuits, which is affected by the total execution time of the circuit and gate errors [22].

In the current work, we propose a solution for qubit mapping in sparse architectures, which is based on binary integer programming. The mapping between logical and physical qubits can be expressed as a matrix whose entries are either 0 or 1. Then the behavior of a **SWAP** gate is equivalent to an elementary row transformation on the mapping matrix, which exchanges two rows of the matrix. The constraints and objective function of qubit mapping can be encoded into a binary integer programming (BIP) problem. As the scale of quantum circuits increases, it is difficult to directly solve the BIP problem in an acceptable amount of time. Therefore, we slice the quantum circuit into smaller ones step by step until sub-circuits are produced that conform to certain patterns. In order to improve the scalability, QMBIP employs other techniques such as path variables and isomorphic pruning to reduce the search space. Isomorphic pruning is a look-ahead mechanism driven by the structure of sliced sub-circuits for the benchmarks. We compare QMBIP with typical qubit mapping algorithms such as SABRE and SATMAP on some sparse architectures including those used in IBM's chips *ibmq_sydney*, *ibmq_manhattan*, *ibmq_singapore* and a dense one *ibmq_tokyo*. Experiments show that QMBIP has a better performance in terms of fidelity. The main contributions of this work are as follows.

- We first consider the qubit mapping problem in sparse architectures and propose a way of slicing quantum circuits by taking into account the sparsity of target architectures.
- We propose an initial mapping algorithm considering the first interactions between any two qubits and encode the constraints and the objective function of qubit mapping into a BIP problem by matrix transformation.
- We use other techniques including path variables and isomorphic pruning to reduce the search space. Our approach turns out to be effective in solving the qubit mapping problem in sparse architectures while maintaining the fidelity of the compiled quantum circuits at a reasonably high level.

The rest of this article is organized as follows. Section 2 reviews basic notions and notations of quantum computing. Section 3 introduces the problem of qubit mapping and the detailed solution. Section 4 uses other techniques for pruning and optimizing. Section 5 reports the experimental results. We conclude in the last section and discuss possible future work.

2 Preliminaries

In this section, we introduce some notions and notations of quantum computing. Let $\mathbb{N}$ and $\mathbb{C}$ denote the set of all natural 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 in the state $|\mathbf{0}\rangle$ with probability $|a|^2$ and in the state $|\mathbf{1}\rangle$ with probability $|b|^2$. In a quantum circuit, each line represents a wire. The wire does not necessarily correspond to a physical wire but may represent the passage of time or a physical particle that moves from one location to another through space. The scale of a quantum circuit is the number of 2-qubit gates. The interested reader can find more details of these gates from the standard textbook [23].

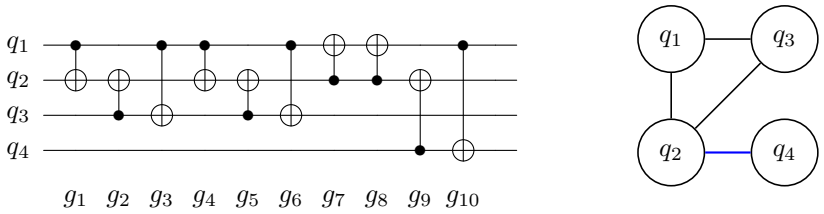


Fig. 1 (a) A quantum circuit; (b) The first interactions between any two qubits

The architecture of a quantum chip called a coupling graph is denoted by $G = (Q, E)$, where Q is a set of physical qubits and E is an edge set between physical qubits. The letter q (resp. Q) denotes a logical (resp. physical) qubit set, where $q = \{q_i\}_{i \in I}$ and $Q = \{Q_j\}_{j \in J}$ for some index sets I and J. A path $Q_1 \to Q_2 \to \ldots \to Q_n$ consists of a set of transformations $Q_i \to Q_{i+1}$, $(1 \leq i \leq n-1)$ with each transformation denoting a **SWAP** gate. A **CX** gate is a pair $g = (q_c, q_t)$ for $c \neq t$ and $c, t \in I$, where q_c is the control qubit and q_t is the target qubit. The mapping between the physical qubits and the logical qubits is a $|Q| \times |q|$-dimensional matrix $\boldsymbol{M}$, where each row (resp. column) corresponds to a physical (resp. logical) qubits. The entry $\boldsymbol{M}_{j,i} = 1$ means that the logical qubit q_i is mapped to the physical qubit Q_j, otherwise $\boldsymbol{M}_{j,i} = 0$, satisfying $\| \boldsymbol{M}_i \|_1 = 1$. The symbol $\| \cdot \|_1$ denotes the 1-norm of a vector. The set $R(q_i) = \{Q_j \mid \boldsymbol{M}_{j,i} \neq 0\}$ consists of the candidate physical qubits mapped by logical qubit q_i. The symbol $R(g)$ denotes a set $\{(Q_1, Q_2) \mid Q_1 \in R(q_c), Q_2 \in R(q_t), \text{and } Q_1 \neq Q_2\}$, which contains all the pairs of candidate physical qubits for the **CX** gate g. The first interaction represents the first **CX** gate acting on two qubits. In Fig. 1 (b), the black edges $(q_1, q_2), (q_1, q_3), (q_2, q_3)$ correspond to the **CX** gate g_1, g_3, g_2 in Fig. 1 (a), respectively, which are in the first slice. The blue edge (q_2, q_4) corresponds to the gate g_9. Fig. 2 shows the architecture of the 27-qubit quantum chip *ibmq_sydney*, where the nodes represent the physical qubits and the edges represent the interactions between

two physical qubits. The label on each edge is the fidelity of a **CX** gate operating on the endpoints. The fidelity of a quantum circuit is the product of the fidelities of all the gates.

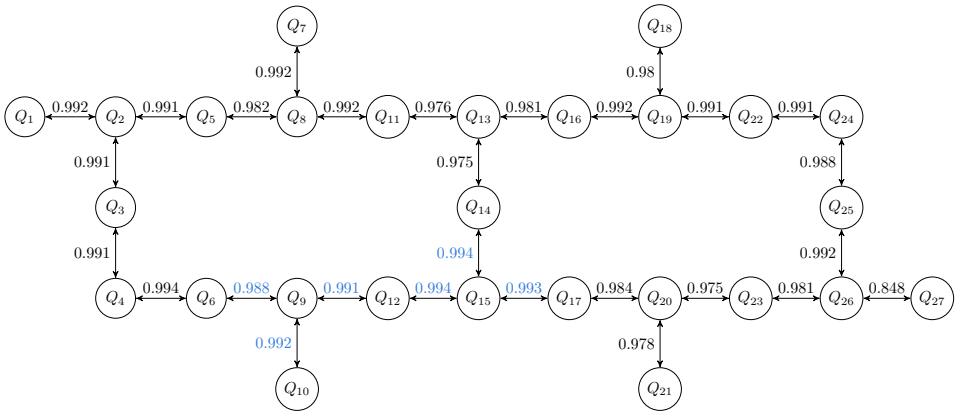


Fig. 2 The architecture of the 27-qubit quantum chip *ibmq_sydney*

The general form of a binary integer programming problem is as follows,

$$\text{minimize} \sum_{i=1}^{n} c_i x_i \tag{1}$$

$$\text{such that: } \sum_{i=1}^{n} a_i x_i = b \text{ and } \bigwedge_{i=1}^{n} (x_i \in \{0,1\}) \tag{2}$$

where the constant coefficients a_i, c_i, b belong to $[0,1]$, (1) is the objective function and (2) is the constraint. To encode the constraints of a slice into a dependency chain of variables, we define a boolean function,

$$f(x_1, x_2, \ldots, x_k) = x_1 \wedge x_2 \wedge \ldots \wedge x_k$$

where $x_1, x_2, \ldots, x_k \in \{0,1\}$ are binary variables. Furthermore, when the input consists of matrices and constants, the general form of f is as follows,

$f(\boldsymbol{X}^1, \ldots, \boldsymbol{X}^l, \tilde{c}, x_1, \ldots, x_k) = \left[f(\boldsymbol{X}^1_{ij}, \ldots, \boldsymbol{X}^l_{ij}, \tilde{c}, x_1, \ldots, x_k,)\right]_{ij}$

where $\tilde{c} = c_1, c_2, \ldots, c_n \in \{0,1\}$ are constants and $\boldsymbol{X}^1, \ldots, \boldsymbol{X}^l \in \{0,1\}^{|Q|\times|Q|}$. Note that the boolean constraints can be linearized.

3 Qubit Mapping

The qubit mapping problem mainly consists of initial mapping and qubit mapping adjustment, which is defined as follows.

Definition 3.1 (The Qubit Mapping Problem [5]). **Input:** a coupling graph $G = (Q, E)$ and a list $\Psi = (q \times q)^n, n \geq 1$, of n control relations between logical qubits q, an integer $K_C \geq 0$, a list of allowed quantum transformations θ, and a function $C : \theta \to \mathbb{N}$ that gives the cost to implement each transformation. **Output:** Yes, if we can produce a version of Ψ that complies with G with transformations whose total cost does not exceed K_C.

3.1 Slicing

The triangle pattern is our criterion for slicing in the topology of a quantum circuit. The reasons are as follows.

- The time cost of solving a BIP problem encoded by the whole circuit is mainly affected by the scale of quantum circuits and target architectures.
- Compiling a quantum circuit on a sparse architecture typically needs to insert some **SWAP** gates as soon as the topology of the **CX** gates forms a triangle.
- There are many consecutive slices with isomorphic interactions on the same qubit set, called similar slices, which are the cornerstone of the look-ahead mechanism.

Example 3.1. *The topology of the quantum circuit in Fig. 1 (a) is shown in Fig. 3. The green nodes represent incoming wire nodes and the red ones represent outgoing wire nodes. The remaining nodes are 2-qubit gates. The label on a directed edge is the operation qubit of the successor node. The in-degree of each gate is equal to the out-degree. We slice the circuit topology based on the triangles marked by the blue lines in Fig. 3. The set of slices is* $\{\{g_1, g_2, g_3\}, \{g_4, g_5, g_6\}, \{g_7, g_8, g_9, g_{10}\}\}$. *Note that the gates* g_7 *and* g_8 *have the same operational qubits, which are sliced into the third slice.*

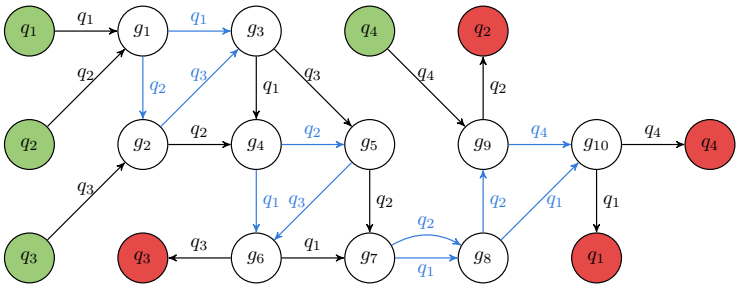


Fig. 3 The topology of the quantum circuit in Fig. 1 (a)

3.2 Initial Mapping

For different physical architectures, there are different initial mapping methods. For example, weighted dependence graph [5] and subgraph isomorphism of the whole quantum circuit [8] are related to the degree of nodes, which are suitable for dense architectures such as *ibmq_tokyo* with an average qubit connectivity of 3.9. In sparse architectures with an average qubit connectivity of 2.1, the initial mapping may be

frequently changed during the process of circuit compilation, even though the interactions of all gates are considered in the initial mapping. Thus, we only consider the first interactions between any two qubits for constructing an initial mapping. First, we find the set of maximal isomorphic subgraphs between the interactions and the target architecture. Then, the matched isomorphic subgraphs are filtered by the sum of the fidelities of each matched physical qubit and its neighbors. Here, we take the sum of those fidelities, which intuitively mimics the degrees of the relevant nodes in the coupling graph. Finally, the remaining nodes are mapped to the physical nodes with the closest degree on the coupling graph.

Algorithm 1 Constructing the initial mapping

Input: The sliced quantum circuit C, the edge set E of the target architecture.
Output: A qubit mapping matrix $\boldsymbol{M}$.

```
G_c ← compute the first interactions between any two qubits in quantum circuit C;
G_e ← compute a maximal isomorphic mapping between G_c and E, filtered by the
    maximum sum of the fidelities of each matched physical qubit and its neighbors;
for all the unmapped logical qubit q do
    N_q ← compute the mapped physical qubits of the neighbors of q in the
    interactions G_c;
    N_p ← compute the unmapped neighbors of the physical qubits in N_q;
    G_e ← G_e ∪ {q ↔ n | n ∈ N_p and with the closest degree to q};
end for
M ← convert the mapping G_e to a 0–1 matrix;
return M;
```

Example 3.2. *Fig. 1 (b) shows the interactions of the first slice* $\{(q_2, q_1), (q_3, q_2), (q_1, q_3)\}$ *interacting with the remaining node* q_4*, where the blue line is the first interaction* (q_4, q_2) *between the remaining qubit* q_4 *and the first slice. There are some maximum common subgraphs* $\{\{q_1 \leftrightarrow Q_1, q_2 \leftrightarrow Q_2, q_4 \leftrightarrow Q_3\}, \{q_1 \leftrightarrow Q_2, q_2 \leftrightarrow Q_3, q_4 \leftrightarrow Q_4\}, \cdots\}$ *between the interactions and the target architecture ibmq_sydney. We use the largest sum of degrees to filter out the maximum common subgraphs with the highest fidelity. The sum of fidelities of the partial mapping* $\{q_1 \leftrightarrow Q_9, q_2 \leftrightarrow Q_{12}, q_4 \leftrightarrow Q_{15}\}$ *is* $2 \times (0.988 + 0.992 + 0.991 + 0.991 + 0.994 + 0.994 + 0.994 + 0.993) = 15.874$*, which is the sum of each label of mapped nodes* $\{Q_9, Q_{12}, Q_{15}\}$ *in Fig. 2. The qubit* q_3 *is still not mapped, which interacts with the mapped qubits* q_1 *and* q_2*. Thus the qubit* q_3 *should be mapped to one of the unmapped neighbors* $\{Q_6, Q_{10}\}$ *of physical qubits* $R(q_1) \cup R(q_2) = \{Q_9, Q_{12}\}$*. The qubit* q_3 *is mapped to the physical qubit* Q_6 *with the closest degree. The initial mapping becomes* $\{q_1 \leftrightarrow Q_9, q_2 \leftrightarrow Q_{12}, q_3 \leftrightarrow Q_6, q_4 \leftrightarrow Q_{15}\}$.

3.3 Constriants

The constraints of qubit mapping refer to the injective mapping from logical qubits to physical ones and the logical gates are executable in physical devices. The executable constraints focus on the interactions between logical gates and the interactions between physical qubits.

3.3.1 Injective Constraint

The mapping from logical qubits to physical ones is injective in which the sum of each column of the mapping matrix $\boldsymbol{M}$ is 1, and the sum of each row is not greater than 1, i. e.,

$$\bigwedge_{i=1}^{|q|}\sum_{j=1}^{|Q|}\boldsymbol{M}_{j,i}=1 \text{ and } \bigwedge_{j=1}^{|Q|}\sum_{i=1}^{|q|}\boldsymbol{M}_{j,i}\leq 1.$$

Suppose a **SWAP** gate acts on a pair of qubits (Q_m, Q_n). Its effect is to exchange the m-th and n-th rows of matrix $\boldsymbol{M}$. Based on this observation, we use a $|Q| \times |Q|$-dimensional elementary matrix to describe a **SWAP** gate.

Example 3.3. *We continue to consider the Example 3.2, focusing on the effect of a* **SWAP** *gate on the initial mapping* $\{q_1 \leftrightarrow Q_9, q_2 \leftrightarrow Q_{12}, q_3 \leftrightarrow Q_6, q_4 \leftrightarrow Q_{15}\}$. *A* **SWAP** *gate acts on the qubit* Q_9 *and* Q_{12}, *which is reflected by the initial mapping matrix* $\boldsymbol{M}_{27\times 4}$ *multiplied by an elementary matrix* $\boldsymbol{P}_{27\times 27}$,

$$\boldsymbol{P}=\begin{array}{c} \\ 6\\7\\8\\9\\10\\11\\12\\13\\14\\15\\ \end{array}\!\!\begin{array}{c}\begin{array}{ccccc} 9 & 10 & 11 & 12 & \end{array}\\ \begin{pmatrix} \ddots & \vdots & \vdots & \vdots & \vdots & \ddots \\ \cdots & 0 & 0 & 0 & 0 & \cdots \\ \cdots & 0 & 0 & 0 & 0 & \cdots \\ \cdots & 0 & 0 & 0 & 0 & \cdots \\ \cdots & 0 & 0 & 0 & 1 & \cdots \\ \cdots & 0 & 1 & 0 & 0 & \cdots \\ \cdots & 0 & 0 & 1 & 0 & \cdots \\ \cdots & 1 & 0 & 0 & 0 & \cdots \\ \cdots & 0 & 0 & 0 & 0 & \cdots \\ \cdots & 0 & 0 & 0 & 0 & \cdots \\ \cdots & 0 & 0 & 0 & 0 & \cdots \\ \ddots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}\end{array},\boldsymbol{M}=\begin{array}{c} \\ 6\\7\\8\\9\\10\\11\\12\\13\\14\\15\\ \end{array}\!\!\begin{array}{c}\begin{array}{cccc} 1 & 2 & 3 & 4 \end{array}\\ \begin{pmatrix} \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}\end{array},\boldsymbol{PM}=\begin{array}{c} \\ 6\\7\\8\\9\\10\\11\\12\\13\\14\\15\\ \end{array}\!\!\begin{array}{c}\begin{array}{cccc} 1 & 2 & 3 & 4 \end{array}\\ \begin{pmatrix} \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}\end{array}.$$

3.3.2 Executable Constraint

Note that 2-qubit gates can only be executed on two adjacent physical qubits and a sequence of **SWAP** gates can exchange the state of a qubit to any connected physical qubit. We use the gray code algorithm [24] to compute the set P of shortest paths that can exchange the states of a qubit pair $(Q_m, Q_n) \in R(g)$ to any target one $(Q_s, Q_r) \in E$ filtered by the fidelities, where $g = (q_c, q_t)$ and $1 \leq m, n, s, r \leq |Q|$. Each path $P_k \in P$ can be represented by a matrix $\boldsymbol{P}_k$, which is the product of a sequence of elementary matrices $\boldsymbol{P}_k = \boldsymbol{P}_{k,|P_k|} \cdots \boldsymbol{P}_{k,2}\boldsymbol{P}_{k,1}$, that acts on the mapping matrix $\boldsymbol{M}$ and results in a mapping matrix $\boldsymbol{M}'_k$ after the adjustments,

$$\boldsymbol{M}'_k = \boldsymbol{P}_k\boldsymbol{M}. \tag{3}$$

Each shortest path is considered as an option. We use a binary variable $x \in \{0,1\}$ to indicate that either the path is chosen ($x=1$) or not ($x=0$). The path $P_k \in P$ moves the state of qubits Q_m, Q_n to the qubits Q_s, Q_r, which is indicated by the variable $x_k \in \{0,1\}$, satisfying $\| \boldsymbol{x} \|_1 = 1$, for $\boldsymbol{x} = [x_1, \ldots, x_{|P|}]^\top \in \{0,1\}^{|P|}$. Whether the path $P_k \in P$ is chosen depends on not only the variable $x_k = 1$, but also the truth value of the current mapped qubit pair in the mapping matrix, i. e. $\boldsymbol{M}_{m,c} = 1$ and $\boldsymbol{M}_{n,t} = 1$. In summary, the path variable v_k is defined as

$$v_k = f(\boldsymbol{M}_{m,c}, \boldsymbol{M}_{n,t}, x_k). \tag{4}$$

Example 3.4. *For the quantum circuit in Fig. 1 (a), the gate g_1 is executable, while the second gate g_2 is not executable. The current qubit pair set of g_2 is $R(g_2) = \{(Q_6, Q_{12})\}$, and the target one is in the set E. We get the set of shortest paths $P = \{P_1, P_2\}$ filtered by the fidelity limitation 0.956, where $P_1 = \{Q_{12} \to Q_9 \to Q_6\}, P_2 = \{Q_6 \to Q_9 \to Q_{12}\}$. The binary variables x_1 and x_2 represent the paths P_1 and P_2, respectively, satisfying the condition $x_1 + x_2 = 1$. The path variable of P_1 is $v_1 = f(\boldsymbol{M}_{6,3}, \boldsymbol{M}_{12,2}, x_1) = f(1,1,x_1)$, where $\boldsymbol{M}_{6,3} = \boldsymbol{M}_{12,2} = 1$. Similarly, the path variable of P_2 is $v_2 = f(\boldsymbol{M}_{6,3}, \boldsymbol{M}_{12,2}, x_2) = f(1,1,x_2)$. If the constraints $x_1 = 0$ and $x_2 = 1$ hold then we also have $v_1 = 0$ and $v_2 = 1$, which means that the state of qubit pair (Q_6, Q_{12}) is swapped to the qubit pair (Q_9, Q_{12}) following the path P_2.*

For each path $P_1, \ldots, P_{|P|}$, there is a sequence of matrix $\boldsymbol{M}'_1, \ldots, \boldsymbol{M}'_{|P|}$ corresponding to it, respectively. The matrix $\boldsymbol{M}''_k$ is combined by the path variable v_k and the matrix $\boldsymbol{M}'_k$ via the function f,

$$\boldsymbol{M}''_k = f(\boldsymbol{M}'_k, v_k) = \begin{cases} \boldsymbol{M}'_k & \text{if } v_k = 1, \\ \boldsymbol{0} & \text{otherwise.} \end{cases} \tag{5}$$

When the value of the path variable v_k is 1, the matrix $\boldsymbol{M}''_k$ is equivalent to $\boldsymbol{M}'_k$, otherwise $\boldsymbol{M}''_k$ is set to be $\boldsymbol{0}$. The matrices $\boldsymbol{M}''_1, ..., \boldsymbol{M}''_{|P|}$ are gathered as the mapping matrix $\boldsymbol{M}'' = \sum_{k=1}^{|P|} \boldsymbol{M}''_k$. Note that all of the path variables satisfy $\| \boldsymbol{v} \|_1 = 1$, where $\boldsymbol{v} = [v_1, \ldots, v_{|P|}]^\top \in \{0,1\}^{|P|}$.

Example 3.5. *Now let us continue to consider the Example 3.4. We only encode the executable constraint of the gates. For the gate g_2, the paths P_1 and P_2 can be represented by the elementary row transformations $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$, respectively,*

$$\boldsymbol{P}_1 = \begin{array}{c} \\ 9 \\ 10 \\ 11 \\ 12 \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{cccccc} & 9 & 10 & 11 & 12 & \end{array} \\ \begin{pmatrix} \ddots & \vdots & \vdots & \vdots & \vdots & \ddots \\ \cdots & 0 & 0 & 0 & 1 & \cdots \\ \cdots & 0 & 1 & 0 & 0 & \cdots \\ \cdots & 0 & 0 & 1 & 0 & \cdots \\ \cdots & 1 & 0 & 0 & 0 & \cdots \\ \ddots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}_{27\times 27} \end{array}, \boldsymbol{P}_2 = \begin{array}{c} \\ 6 \\ 7 \\ 8 \\ 9 \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{cccccc} & 6 & 7 & 8 & 9 & \end{array} \\ \begin{pmatrix} \ddots & \vdots & \vdots & \vdots & \vdots & \ddots \\ \cdots & 0 & 0 & 0 & 1 & \cdots \\ \cdots & 0 & 1 & 0 & 0 & \cdots \\ \cdots & 0 & 0 & 1 & 0 & \cdots \\ \cdots & 1 & 0 & 0 & 0 & \cdots \\ \ddots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}_{27\times 27} \end{array}.$$

After the elementary row transformation $\boldsymbol{P}_1$ and combining with corresponding path variable v_1, we get the mapping matrices $\boldsymbol{M}_1' = \boldsymbol{P}_1\boldsymbol{M}$ and $\boldsymbol{M}_1'' = f(\boldsymbol{M}_1', v_1)$,

$$\boldsymbol{M}_1' = \begin{array}{c} \\ \vdots \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \\ \vdots \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}_{27\times 4} \end{array}, \boldsymbol{M}_1'' = \begin{array}{c} \\ \vdots \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \\ \vdots \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & f(1,v_1) & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & f(1,v_1) & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ f(1,v_1) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & f(1,v_1) \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}_{27\times 4} \end{array}.$$

Similarly, after the elementary row transformation $\boldsymbol{P}_2$ and combining with corresponding path variable v_2, we get the matrices $\boldsymbol{M}_2'$ and $\boldsymbol{M}_2'' = f(\boldsymbol{M}_2', v_2)$, respectively,

$$\boldsymbol{M}_2' = \begin{array}{c} \\ \vdots \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \\ \vdots \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}_{27\times 4} \end{array}, \boldsymbol{M}_2'' = \begin{array}{c} \\ \vdots \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \\ \vdots \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} \vdots & \vdots & \vdots & \vdots \\ f(1,v_2) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & f(1,v_2) & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & f(1,v_2) & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & f(1,v_2) \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}_{27\times 4} \end{array}.$$

Finally, we get the mapping matrix $\boldsymbol{M}''$ by combining with the elementary row transformations $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$,

$$
\boldsymbol{M}'' = \boldsymbol{M}_1'' + \boldsymbol{M}_2'' = \begin{array}{c} \\ \vdots \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \\ \vdots \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} \vdots & \vdots & \vdots & \vdots \\ f(1,v_2) & 0 & f(1,v_1) & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & f(1,v_1) & f(1,v_2) & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ f(1,v_1) & f(1,v_2) & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & f(1,v_1)+f(1,v_2) \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}_{27\times 4} \end{array}.
$$

where the entry $f(1,v_1)+f(1,v_2)$ of matrix $\boldsymbol{M}''$ equals to 1, since either $v_1 = 1$ or $v_2 = 1$ holds. So are the two nonzero entries $f(1,v_1)$ and $f(1,v_2)$ in the columns of matrix $\boldsymbol{M}''$. Thus, the constraints for the quantum gate g_2 are as follows,

$$
\begin{cases} x_1 + x_2 = 1, \\ v_1 + v_2 = 1, \\ x_1, x_2, v_1, v_2 \in \{0,1\}. \end{cases}
$$

For the third gate $g_3 = (q_1, q_3)$, the qubits q_1 and q_3 are mapped to $R(q_1) = \{Q_6, Q_{12}\}$ and $R(q_3) = \{Q_6, Q_9\}$, respectively. The set of currently mapped qubit pairs is $R(g_3) = \{(Q_6,Q_9),(Q_{12},Q_6),(Q_{12},Q_9)\}$. Then there are four paths $P = \{P_3 = \{Q_6 \to Q_9\}, P_4 = \{Q_{12} \to Q_9 \to Q_6\}, P_5 = \{Q_6 \to Q_9 \to Q_{12}\}, P_6 = \{Q_{12} \to Q_9\}\}$ filtered by the fidelity limitation 0.956. The path variables are as follows,

$$
\begin{aligned}
v_3 &= f(\boldsymbol{M}_{6,1}, \boldsymbol{M}_{9,3}, x_3) = f(f(1,v_2), f(1,v_2), x_3), \\
v_4 &= f(\boldsymbol{M}_{12,1}, \boldsymbol{M}_{6,3}, x_4) = f(f(1,v_1), f(1,v_1), x_4), \\
v_5 &= f(\boldsymbol{M}_{12,1}, \boldsymbol{M}_{6,3}, x_5) = f(f(1,v_1), f(1,v_1), x_5), \\
v_6 &= f(\boldsymbol{M}_{12,1}, \boldsymbol{M}_{9,3}, x_6) = f(f(1,v_1), f(1,v_2), x_6) = 0.
\end{aligned} \tag{6}
$$

Finally, the constraints of the gate g_3 are as follows,

$$
\begin{cases} x_3 + x_4 + x_5 + x_6 = 1, \\ v_3 + v_4 + v_5 + v_6 = 1, \\ x_3, x_4, x_5, x_6, v_3, v_4, v_5, v_6 \in \{0,1\}. \end{cases}
$$

3.4 Objective Function

The most accurate metric for evaluating a compiled circuit is the fidelity of the generated circuit, which is affected primarily by the error rate of the gates in the circuit and the error caused by decoherence. The objective function considers only the fidelity of each path,

$$\text{maximize} \sum_{i=1}^{n} \sum_{k=1}^{|P_i|} w_k v_k,$$

where n is the number of gates in a slice and $|P_i|$ is the size of the shortest path set for the i-th gate. The weight w_k is the fidelity of the path indicated by the variable v_k. By simply modifying the label of each edge on the architecture to a constant e, where $0 < e < 1$, the objective function is extended to the least number of 2-qubit gates inserted. We summarize the encoding procedure in Algorithm 2.

Algorithm 2 Encoding into binary integer programming

Input: A slice of the quantum circuit G, a qubit mapping matrix $\boldsymbol{M}$, the edge set E of the target architecture and the fidelity limitation W.
Output: A constraint set T and the objective function F.

$F \leftarrow 0$ and $T \leftarrow \emptyset$;
for all g in G **do**
  if g is a 2-qubit gate **then**
    $\boldsymbol{M}'' \leftarrow \boldsymbol{0}$;
    $P \leftarrow$ compute the set of the shortest paths from each qubit pair $(Q_m, Q_n) \in R(g)$ to any edge in E filtered by the fidelity limitation W;
    let $\boldsymbol{x} = [x_1, \ldots, x_{|P|}]^\top$ be a vector of binary variables and $\boldsymbol{v} = [v_1, \ldots, v_{|P|}]^\top$ be a vector of path variables for each path $P_k \in P$ as defined in (4);
    for $k \leftarrow 1$ to $|P|$ **do**
      $\boldsymbol{M}'_k \leftarrow \boldsymbol{M}$;
      $Q_m, Q_n \leftarrow$ the currently mapped qubit pairs of P_k;
      let w be the fidelity of the path P_k;
      compute the mapping matrix $\boldsymbol{M}'_k$ along the path P_k by (3) and obtain $\boldsymbol{M}''_k$ by combining $\boldsymbol{M}'_k$ and path variable v_k as defined in (5);
      $F \leftarrow F + w \cdot v_k$;
      $\boldsymbol{M}'' \leftarrow \boldsymbol{M}'' + \boldsymbol{M}''_k$;
    end for
    $\boldsymbol{M} \leftarrow \boldsymbol{M}''$;
    $T \leftarrow T \cup \{\|\boldsymbol{v}\|_1 = 1, \|\boldsymbol{x}\|_1 = 1\}$;
  end if
end for
return T and F;

Example 3.6. *Continuing to the Example 3.5, we consider encoding the objective function. For each path variable, the weight is the fidelity of that characterized path.*

Thus, the objective function for the first slice is as follows,

$$F = 0.962v_1 + 0.956v_2 + 0.988v_3 + 0.962v_4 + 0.956v_5 + 0.991v_6.$$

The solution of the first slice is $x_1 = 0, x_2$–$x_3 = 1, x_4$–$x_6 = 0$. *That is a* **SWAP** *gate acts on the qubits* Q_6 *and* Q_9 *inserted before the gate* g_2. *Similarly, the constraints and objective of each slice are encoded into a BIP problem and solved. Then the compiled quantum circuit is as shown in Fig. 4.*

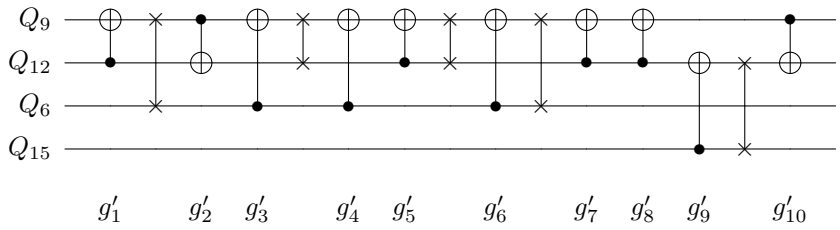


Fig. 4 The compiled quantum circuit

3.5 Complexity

The time complexity of our compilation is determined by the pruning strength. For a quantum gate g, there are $|R(g)| \times |E|$ search paths. The times complexity of encoding a quantum circuit C is $|R(g)| \times |E| \times |C|$, where $|C|$ is the scale of quantum circuit C.

4 Efficiency

4.1 Pruning

In our approach, the number of variables is related to the size of the shortest path set. In the worst case, there are $|R(g)| \times |E|$ shortest paths for encoding a gate where $|R(g)| = |Q| \times |Q| - 1$. First, we consider pruning the set of target qubit pairs E by isomorphism, called isomorphic pruning, since pruning the currently mapped qubits set $R(g)$ will affect the existing variables. Some heuristics with a look-ahead mechanism consider unexecutable gates in the objective function. We observed existing similar slices in quantum circuits, which is the cornerstone of the look-ahead mechanism in our approach. The isomorphic edges between the interactions of the similar slice and the target architecture are considered as the target qubit pairs, which not only satisfies the execution of the gates in that slice but also takes the following similar slices into account, which is a look-ahead mechanism. When the gates in a slice are mapped, the gates in the next similar slice are also mapped in the maximum isomorphic subgraph, which only needs inserting a few **SWAP** gates. The scale of target qubit pairs of a slice reduces from $|G| \times |E|$ to several maximum isomorphic subgraphs, where $|G|$ is

the number of matched gates in a slice. Other gates still use the set E as the set of target qubit pairs and the paths $P_k \in P$ that have a low fidelity will be filtered out.

Example 4.1. *In Example 3.6, the first and second slices are similar, since their interactions on the same qubit set are isomorphic. The maximum isomorphic subgraph of the first slice filtered by the maximum fidelity remains* $\{(Q_9, Q_{12}), (Q_{12}, Q_{15})\}$*, which is also the maximum isomorphic subgraph of second slice* $\{g_4, g_5, g_6\}$*. Then, the path set for the second gate is* $\{\{Q_6 \to Q_9 \to Q_{12}\}\}$ *and the path set for the third gate is* $\{\{Q_6 \to Q_9\}\}$*. After isomorphic pruning, there are only 2 path variables for encoding the first slice. Compared with the encoding method without isomorphic pruning in Example 3.6, isomorphic pruning uses 4 fewer path variables.*

4.2 Optimization

The optimization of qubit mapping is in three aspects, look-ahead in the initial mapping, gate cancellation and rearrangement of 1-qubit gates. In the previous work, Li *et al.* [7] reverses the whole circuit to optimize the initial mapping. Here we make use of this idea for our optimization. Let $C_{1:n} = C_1 C_2 \dots C_n$ denote the first n slices in a quantum circuit C, where C_i, $1 \le i \le n$ is a slice. After compiling the fragment $C_{1:n}$, the mapping is denoted by $\boldsymbol{M'}$, which is used as the initial mapping for compiling the inverse of the fragment $C_{1:n}^{-1}$. Finally, we get the mapping $\boldsymbol{M''}$, which is used as a new initial mapping. Our approach looks ahead to one slice for constructing the initial mapping, i. e. $n = 2$. Furthermore, the inserted **SWAP** gates may produce gate cancellation with the existing **CX** gates in the quantum circuit. Thus we consider a simple optimization of the compiled circuit, which contains gate cancellation and commutation gates [25]. The position of the **SWAP** gates that are inserted affects the fidelity of the neighboring 1-qubit gates. Our approach rearranges **SWAP** gates and 1-qubit gates to improve the fidelity of 1-qubit gates.

5 Experimental Evaluation

5.1 Implementation and Benchmarks

QMBIP is implemented in Matlab 2019a with a solver Gurobi and Python 3.7. All the experiments are conducted on a Ubuntu machine with a 2.2GHz CPU and 64G memory. Considering the factors of solving time and local optima, we use some functions of RevLib [26] with no more than 300 2-qubit gates as benchmarks, which have also been adopted in several related work. We compare QMBIP with the latest constrained-based algorithm SATMAP [14], which is based on maximum satisfiability and the industry-recognized excellent heuristic algorithm SABRE [7]. Table 2 lists the number and time cost of benchmarks successfully solved by the approaches SABRE, SATMAP and QMBIP on three sparse architectures and a dense one. The compilation time for a single instance is within 24 hours. The data of those target architectures is reported from Qiskit[4]. Figs. 5–9 more intuitively show the performance of the three methods on the benchmark and the blue markers indicate the performance of QMBIP, the green indicates SATMAP and the red indicates SABRE.

[4] https://Qiskit.org, accessed 17 December 2023.

Table 2 Number of benchmarks and total time cost successfully solved by SABRE, SATMAP and QMBIP within 24 hours

			SABRE		SATMAP		QMBIP	
#qubit	architecture	#benchmark	#solved	time (s)	#solved	time (s)	#solved	time (s)
27	*sydney*	85	85	460	69	272390	85	129283
65	*manhattan*	63	63	882	38	221175	63	246437
20	*singapore*	85	85	160	74	299152	85	119970
20	*tokyo*	85	85	66	85	77442	85	247403

5.2 Research questions

Our experiments focus on answering the following three questions.

1. How does QMBIP compare to constraint-based and heuristic approaches?
2. How does QMBIP perform with another objective?
3. What is the trade-off between scalability and optimality?

1. How does QMBIP compare to constraint-based techniques and heuristic approaches?

First, we compare the fidelity of QMBIP with SATMAP and SABRE. In the first row of Table 2, we list the number of instances successfully compiled by SABRE, SATMAP and QMBIP among 85 instances on the architecture *ibmq_sydney* and the total time cost in seconds. SATMAP successfully compiled 69 instances in 272390s, while SABRE and QMBIP successfully compiled 85 instances in less time. In Fig. 5, there is a more intuitive view of the fidelity for each instance, where the x-axis represents the number of 2-qubit gates in each instance and the y-axis represents the fidelity of the compiled circuits or the compilation time. The square markers indicate the difference in fidelity and the line charts indicate the time cost among SABRE, QMBIP and SATMAP for that instance. As the scale of the circuit increases, the fidelity of the compiled circuit gradually decreases. The fidelity approximates 0 when the scale of the quantum circuit is more than 200. In Fig. 5, we can see that the fidelity of QMBIP in blue is almost the highest since SABRE and SATMAP consider the number of **SWAP** gates inserted and do not take into account the differences between physical qubits. Comparing the fidelity of the compiled instances of the three approaches, all instances compiled by QMBIP outperform SABRE and SATMAP by 53.9% and 46.8% on average, respectively. The heuristic algorithm SABRE uses the least amount of time since constraint-based solving algorithms are affected by the scale of quantum circuits and the connectivity of the target architecture. QMBIP uses 52% less time than SATMAP on average and the blue lines are below the green lines in most instances as shown in Fig. 5.

2. How does QMBIP perform with other objectives?

In addition to the fidelity, we also compare the number of 2-qubit gates inserted by QMBIP, SABRE and SATMAP on *ibmq_sydney*. Fig. 6 shows the number of 2-qubit gates inserted compiled by the three approaches in different scales, where the x-axis represents the number of 2-qubit gates in each instance and the y-axis represents the number of 2-qubit gates inserted or the compilation time. The bars indicate the

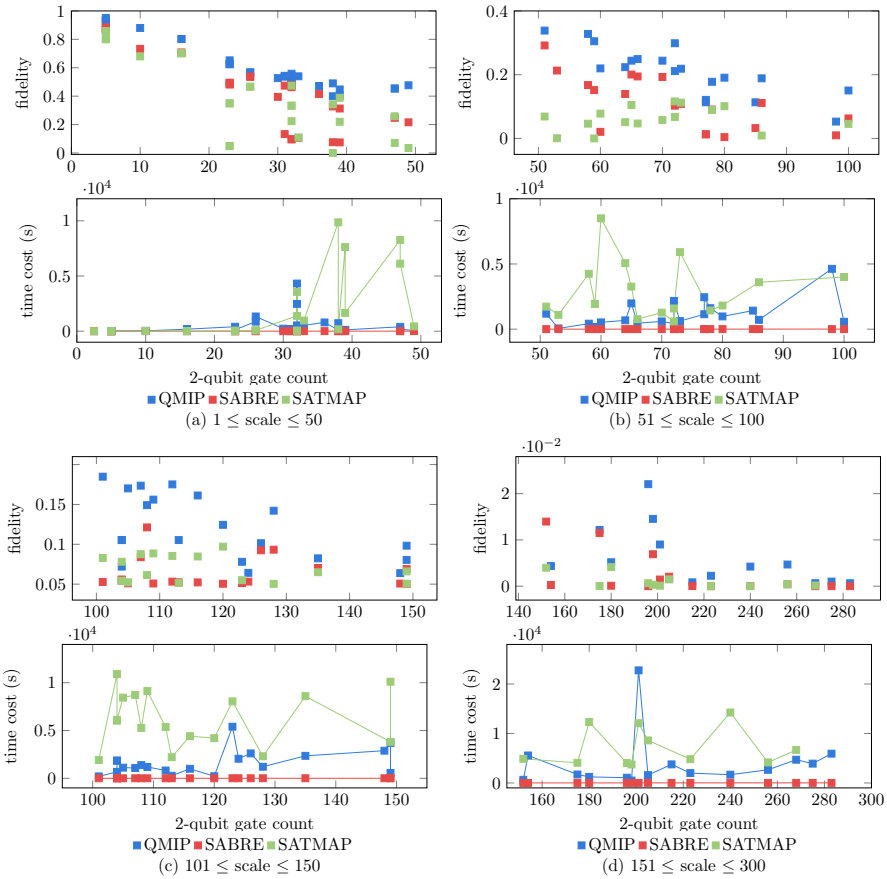


Fig. 5 The fidelity and time cost of each instance while compiling on *ibmq_sydney*

number of 2-qubit gates inserted and the line charts indicate the time cost of QMBIP, SATMAP and SABRE. We can clearly see that the number of 2-qubit gates inserted by SABRE is almost the highest in all instances. Within the range of 1–100, the blue bars are the lowest, in addition to the x-coordinate 26 in Fig. 6 (a). Within the range of 101–300, there are 21 out of 33 instances for which QMBIP inserts the fewest 2-qubit gates. In Fig. 6, there is a lack of 13 green bars, which means those instances could not be compiled by SATMAP successfully within 24 hours. On average, QMBIP adds the least number of 2-qubit gates with a reduction of 11 (resp. 10) **CX** gates in comparison with SABRE (resp. SATMAP) for each instance. The heuristic approach SABRE uses the least amount of time and both constraint-based methods have a large fluctuation.

3. What is the trade-off between scalability and optimality?

In addition to experimenting on the architecture *ibmq_sydney*, we experiment on other sparse architectures *ibmq_manhattan*, *ibmq_singapore* and a dense architecture *ibmq_tokyo*. Each row of Table 2 shows the number of solved benchmarks and time

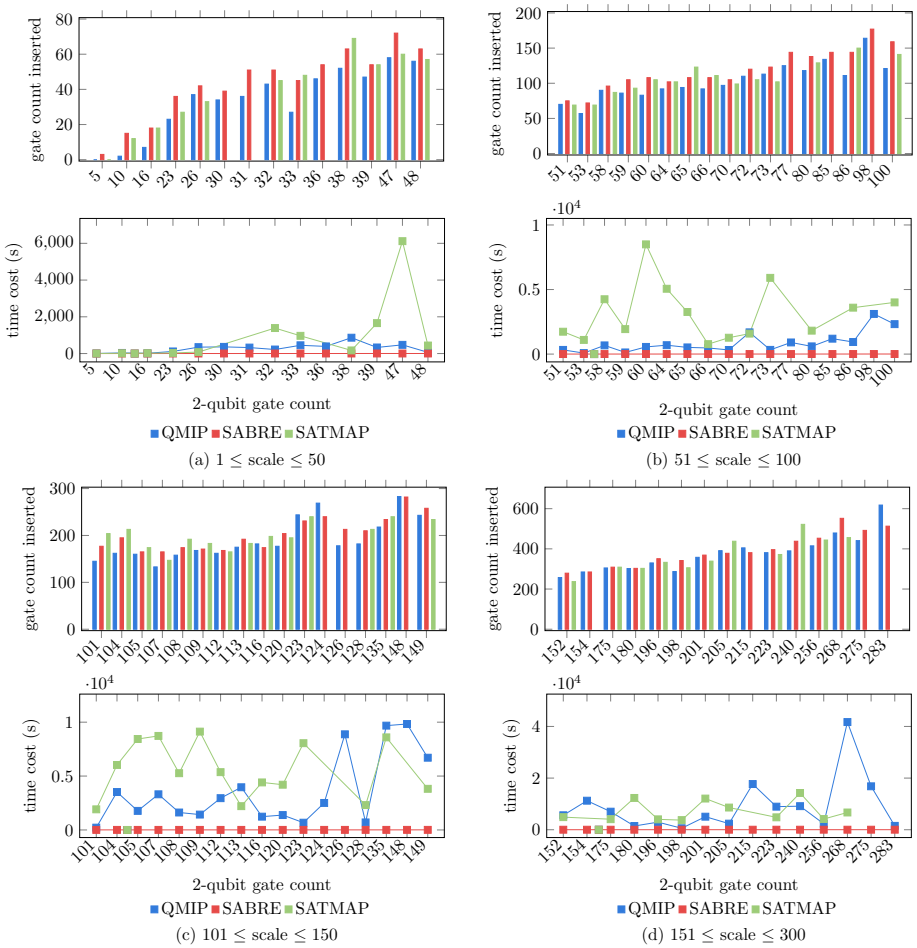


Fig. 6 The number of 2-qubit gates inserted and time cost while compiling on *ibmq_sydney*

costs for different architectures, compiled by QMBIP, SATMAP and SABRE. For the architecture *ibmq_manhattan*, the error rate on some edges is 1, so we compare the number of 2-qubit gates inserted. To show the performance of SABRE, SATMAP and QMBIP on architecture *ibmq_manhattan* more clearly, only one instance with the same scale is retained in the benchmarks. There are a total of 63 benchmark tests in *ibmq_manhattan*, as shown in the second row of Table 2, of which SATMAP only successfully compiled 38 instances. Both *ibmq_singapore* and *ibmq_tokyo* test 85 benchmarks in total, as shown in the third and fourth row of Table 2, respectively.

In Fig. 7, the x-axis represents the number of 2-qubit gates in each instance and the y-axis represents the number of 2-qubit gates inserted or the compilation time. There are some instances that SATMAP can handle on the 27-qubit architecture *ibmq_sydney* but is not able to handle on the 65-qubit architecture *ibmq_manhattan*, such as the

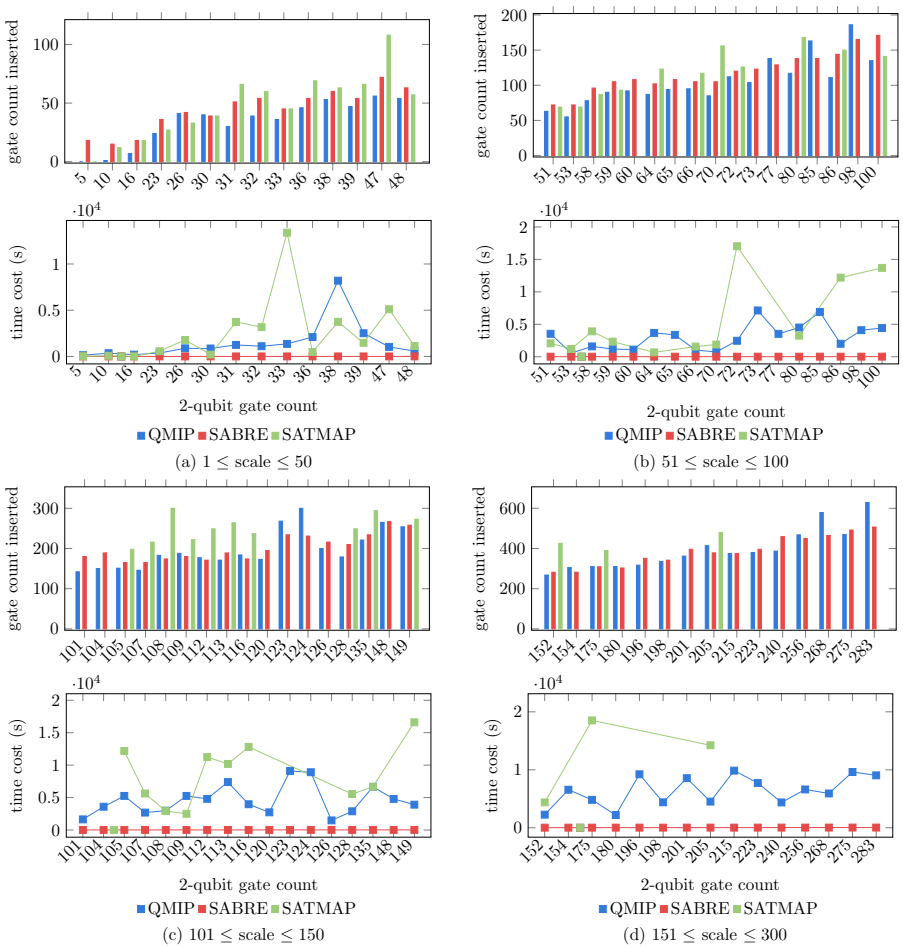


Fig. 7 The number of 2-qubit gates inserted and time cost while compiling on *ibmq_manhattan*

x-coordinates $101, 104, 120, 123, 124, 126, 148$ in Fig. 7 (c). For the instances in Fig. 7 (d), SATMAP has 12 out of 15 circuits unsuccessfully compiled within 24h. It demonstrated that SATMAP is sensitive to changes in circuit scale. As for the compiled circuits, QMBIP has a higher fidelity than SABRE (resp. SATMAP) in 45 out of 63 (resp. 35 out of 38) instances. On the sparse architecture *ibmq_singapore*, there is a similar performance for the fidelity of compiled quantum circuits, as shown in Fig. 8. Comparing the fidelity of the compiled instances by the three approaches, QMBIP is higher than SABRE in 81 out of 85 instances and higher than SATMAP in 68 out of 74 instances. QMBIP also can be extended to dense architectures, such as *ibmq_tokyo*. Fig. 9 shows the fidelity and time cost of each instance of QMBIP, SABRE, and SATMAP compiled in the architecture *ibmq_tokyo*. QMBIP maintains a higher fidelity than SABRE in 60 out of the 85 instances and has a higher fidelity than SATMAP in

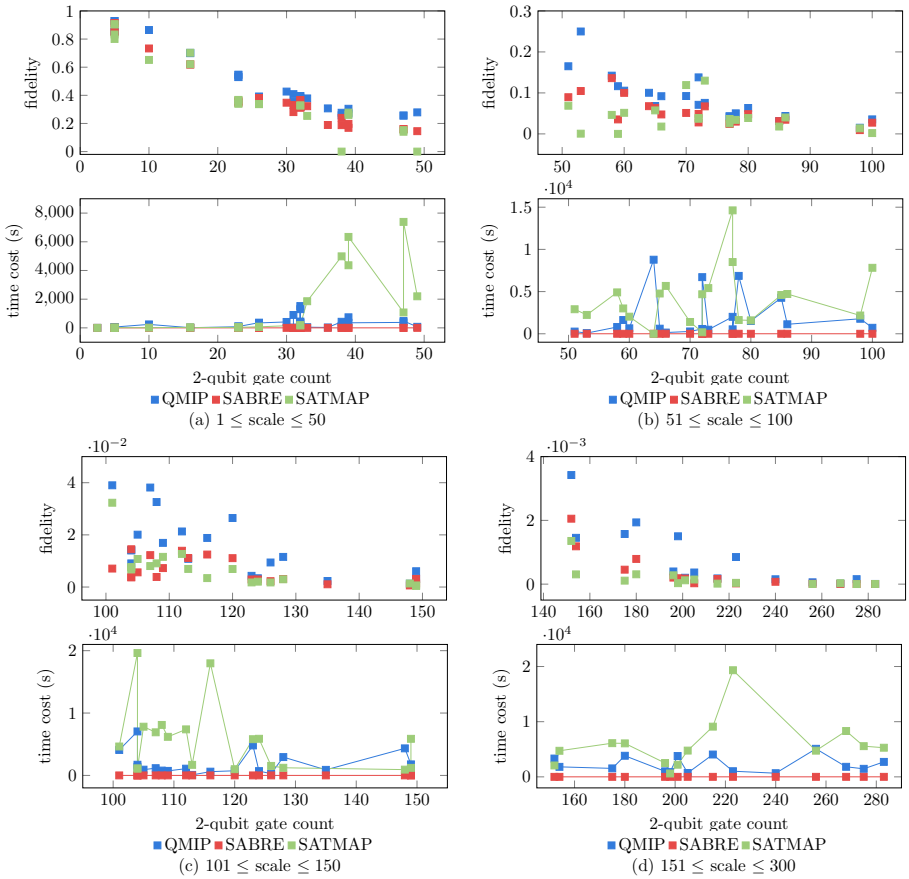


Fig. 8 The fidelity and time cost while compiling on *ibmq_singapore*

39 out of the 85 instances. Thus, QMBIP has better performance in sparse architectures. Comparing the scalability of the three approaches on a 65-qubit architecture, *ibmq_manhattan*, a 27-qubit architecture *ibmq_sydney* and the 20-qubit architecture *ibmq_singapore*, the time cost of SABRE is steadily maintained at a minimum in all of the benchmarks. As the scale of the instances increases, there is a significant increase in the time cost of constraint-based solving approaches including both QMBIP and SATMAP. For different instances, QMBIP and SATMAP have large fluctuations in time cost for obtaining higher fidelities. QMBIP easily adapts to changes in target architectures because of isomorphic pruning, while SATMAP is sensitive to the scale of the target architecture. From Figs. 5–8, we can see that QMBIP outperforms SABRE and SATMAP in terms of fidelity or inserted 2-qubits gates in most instances.

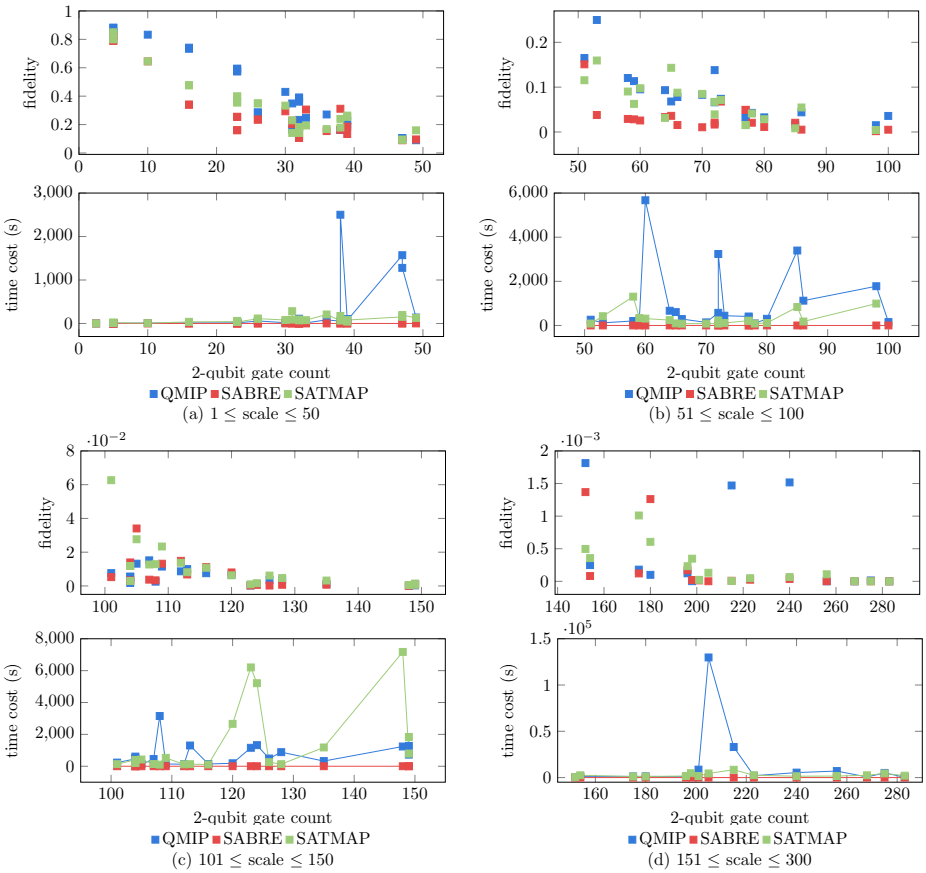


Fig. 9 The fidelity and time cost while compiling on *ibmq_tokyo*

5.3 Discussion

Nowadays the mainstream architecture design direction of quantum chips is to adopt a low connectivity, which is beneficial to double the number of qubits. The constraint-based solving approaches are affected by the scale of circuits and the connectivity of target architectures. QMBIP is designed for sparse architectures and is less affected by the scale of the target architecture. The interactions of the whole quantum circuit are generally strongly connected. We consider the first interactions between any two qubits for constructing the initial mapping, which is adjusted as the 2-qubit gates are executed step by step. The complexity of encoding is exponential in terms of the degrees of chip nodes. We reduce the number of variables from the number of edges to the number of shortest paths and filter them by fidelities. For similar slices, isomorphic pruning effectively reduces the number of variables. The constraint-based solving approaches use significantly more time than the heuristic algorithm, which can be shortened by pruning.

6 Conclusion

In this paper, we have investigated the problem of qubit mapping in sparse architectures. A quantum circuit was sliced step by step according to certain patterns for scalability. We considered the first interactions between any two qubits for constructing the initial mapping. To make a trade-off between the compilation time and the objective, we have proposed to use path variables and isomorphic pruning together with a look-ahead mechanism to reduce the search space. The experiments have validated the improvement in the fidelity of the quantum circuits and the scalability on different scales of architectures compiled by QMBIP. For future work, we plan to improve the efficiency of QMBIP and reduce the effect of local optima in large-scale circuits by more precise pruning.

Acknowledgements. Deng was supported by the National Key R&D Program of China under Grant No. 2023YFA1009403, the National Natural Science Foundation of China under Grant No. 62072176, and the "Digital Silk Road" Shanghai International Joint Lab of Trustworthy Intelligent Software under Grant No. 22510750100.

Declarations

Conflict of interest. The authors declare no competing interests.

References

[1] Benioff, P.: The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines. Journal of Statistical Physics **22**, 563–591 (1980) https://doi.org/10.1007/BF01011339

[2] Debnath, S., Linke, N.M., Figgatt, C., Landsman, K.A., Wright, K., Monroe, C.: Demonstration of a small programmable quantum computer with atomic qubits. Nature **536**(7614), 63–66 (2016) https://doi.org/10.1038/nature18648

[3] Koch, J., Yu, T.M., Gambetta, J., Houck, A.A., Schuster, D.I., Majer, J., Blais, A., Devoret, M.H., Girvin, S.M., Schoelkopf, R.J.: Charge-insensitive qubit design derived from the cooper pair box. Phys. Rev. A **76**, 042319 (2007) https://doi.org/10.1103/PhysRevA.76.042319

[4] Zhong, H.-S., Wang, H., Deng, Y.-H., Chen, M.-C., Peng, L.-C., Luo, Y.-H., Qin, J., Wu, D., Ding, X., Hu, Y., Hu, P., Yang, X.-Y., Zhang, W.-J., Li, H., Li, Y., Jiang, X., Gan, L., Yang, G., You, L., Wang, Z., Li, L., Liu, N.-L., Lu, C.-Y., Pan, J.-W.: Quantum computational advantage using photons. Science **370**(6523), 1460–1463 (2020) https://doi.org/10.1126/science.abe8770

[5] Siraichi, M.Y., Santos, V.F.d., Collange, C., Pereira, F.M.Q.: Qubit allocation. In: Proceedings of the 2018 International Symposium on Code Generation and Optimization. CGO 2018, pp. 113–125. ACM, New York (2018). https://doi.org/10.1145/3168822

[6] Paler, A.: On the influence of initial qubit placement during nisq circuit compilation. In: Feld, S., Linnhoff-Popien, C. (eds.) Quantum Technology and Optimization Problems, pp. 207–217. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-14082-3_18

[7] Li, G., Ding, Y., Xie, Y.: Tackling the qubit mapping problem for NISQ-era quantum devices. In: Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems. ASPLOS '19, pp. 1001–1014. ACM, New York (2019). https://doi.org/10.1145/3297858.3304023

[8] Zhou, X., Li, S., Feng, Y.: Quantum circuit transformation based on simulated annealing and heuristic search. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **39**(12), 4683–4694 (2020) https://doi.org/10.1109/TCAD.2020.2969647

[9] Li, S., Zhou, X., Feng, Y.: Qubit mapping based on subgraph isomorphism and filtered depth-limited search. IEEE Transactions on Computers **70**(11), 1777–1788 (2021) https://doi.org/10.1109/TC.2020.3023247

[10] Venturelli, D., Do, M., Rieffel, E.G., Frank, J.: Temporal planning for compilation of quantum approximate optimization circuits. In: Proceedings of the 26th International Joint Conference on Artificial Intelligence, pp. 4440–4446 (2017). https://doi.org/10.24963/ijcai.2017/620

[11] Bernal, D.E., Booth, K.E.C., Dridi, R., Alghassi, H., Tayur, S.R., Venturelli, D.: Integer programming techniques for minor-embedding in quantum annealers. In: Proceedings of the 17th International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research, pp. 112–129. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58942-4_8

[12] Almeida, A.A.A., Dueck, G.W., Silva, A.C.R.: Finding optimal qubit permutations for IBM's quantum computer architectures. In: Proceedings of the 32nd Symposium on Integrated Circuits and Systems Design. SBCCI '19, pp. 1–6. ACM, Sao Paulo, Brazil (2019). https://doi.org/10.1145/3338852.3339829

[13] Wille, R., Burgholzer, L., Zulehner, A.: Mapping quantum circuits to IBM QX architectures using the minimal number of SWAP and H operations. In: Proceedings of the 56th Annual Design Automation Conference 2019. DAC '19, pp. 1–6. ACM, New York (2019). https://doi.org/10.1145/3316781.3317859

[14] Molavi, A., Xu, A., Diges, M., Pick, L., Tannu, S.S., Albarghouthi, A.: Qubit mapping and routing via maxsat. In: 55th IEEE/ACM International Symposium on Microarchitecture, pp. 1078–1091. IEEE, Chicago, Illinois (2022). https://doi.org/10.1109/MICRO56248.2022.00077

[15] Zulehner, A., Paler, A., Wille, R.: An efficient methodology for mapping quantum

circuits to the IBM QX architectures. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **38**(7), 1226–1236 (2019) https://doi.org/10.1109/TCAD.2018.2846658

[16] Zhu, P., Guan, Z., Cheng, X.: A dynamic look-ahead heuristic for the qubit mapping problem of NISQ computers. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **39**(12), 4721–4735 (2020) https://doi.org/10.1109/TCAD.2020.2970594

[17] Cowtan, A., Dilkes, S., Duncan, R., Krajenbrink, A., Simmons, W., Sivarajah, S.: On the qubit routing problem. In: Proceedings of the 14th Conference on the Theory of Quantum Computation, Communication and Cryptography. LIPIcs, vol. 135, pp. 1–32. Dagstuhl, Germany (2019). https://doi.org/10.4230/LIPIcs.TQC.2019.5

[18] Jiang, H., Deng, Y., Xu, M.: Qubit mapping based on tabu search. Journal of Computer Science and Technology **39**(2), 421–433 (2023). https://jcst.ict.ac.cn/en/article/doi/10.1007/s11390-023-2121-5

[19] 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), Seoul, Korea (South), pp. 167–178 (2021). https://doi.org/10.1109/HPCA51647.2021.00024

[20] Lao, L., Someren, H., Ashraf, I., Almudever, C.G.: Timing and resource-aware mapping of quantum circuits to superconducting processors. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **41**(2), 359–371 (2021) https://doi.org/10.1109/TCAD.2021.3057583

[21] Zhang, C., Hayes, A.B., Qiu, L., Jin, Y., Chen, Y., Zhang, E.Z.: Time-optimal qubit mapping. In: Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. ASPLOS '21, pp. 360–374. ACM, New York (2021). https://doi.org/10.1145/3445814.3446706

[22] Murali, P., Mckay, D.C., Martonosi, M., Javadi-Abhari, A.: Software mitigation of crosstalk on noisy intermediate-scale quantum computers. In: Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems. ASPLOS '20, pp. 1001–1016. ACM, New York (2020). https://doi.org/10.1145/3373376.3378477

[23] Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information: 10th Anniversary Edition, 10th edn. Cambridge University Press, Cambridge (2011)

[24] Gray, F.: Pulse Code Communication, US (1953)

[25] Jiang, H., Li, D., Deng, Y., Xu, M.: A pattern matching-based framework for quantum circuit rewriting. Journal of Computer Science and Technology (2024). https://jcst.ict.ac.cn/en/article/doi/10.1007/s11390-024-2726-3, (in press). Accessed 20 March 2024

[26] Wille, R., Groe, D., Teuber, L., Dueck, G.W., Drechsler, R.: Revlib: An online resource for reversible functions and reversible circuits. In: Proceedings of the 38th International Symposium on Multiple Valued Logic. ISMVL '08, pp. 220–225. IEEE Computer Society, Dallas (2008). https://doi.org/10.1109/ISMVL.2008.43