Quantum Information Processing (2024) 23:200
https://doi.org/10.1007/s11128-024-04425-7

A processor architecture design method for improving reusability of special-purpose superconducting quantum processor

Tian Yang[1] · Weilong Wang[1] · Bo Zhao[1] · Lixin Wang[1] · Xiaodong Ding[1] · Chen Liang[1,2] · Zheng Shan[1]

Received: 30 October 2023 / Accepted: 8 May 2024 / Published online: 23 May 2024

Abstract
Optimizing the architecture of superconducting quantum processors is crucial for improving the efficiency of executing quantum programs. Existing schemes either modify general-purpose architectures, which might lead to an increase in the probability of qubit frequency collisions, or customize special-purpose architectures based on the quantum programs to reduce the gate operations after qubit mapping, but the architectures lack support for the post-mapping gate operations' optimization of multiple programs, which reduce their reusability. In this study, we propose a new processor architecture design method that reduces the average growth of the total post-mapping gate count on multiple quantum programs as well as to reduce the impact of processor architecture on frequency collisions, and thus improve the reusability of special-purpose processor. The main idea is to construct a new architecture by finding maximum common edge subgraph among multiple special-purpose processor archi-

✉ Weilong Wang
wangwl19888@163.com

✉ Zheng Shan
shanzhengzz@163.com

Tian Yang
ly_yangtian@163.com

Bo Zhao
zhaob07@tsinghua.org.cn

Lixin Wang
wlx715@163.com

Xiaodong Ding
dxdlhh@126.com

Chen Liang
lc18937658696@163.com

1 Laboratory for Advanced Computing and Intelligence Engineering, Information Engineering University, Zhengzhou 450001, Henan, China

2 School of Cyber Science and Engineering, Zhengzhou University, Zhengzhou 450002, Henan, China

tectures. To show the effectiveness of our method, we selected quantum programs with different functions covering 9 types of qubit numbers for comparison. Comprehensive simulation results show that the architecture schemes generated by using our method outperform two general-purpose architecture schemes based on the square lattice and the eff-5-freq's special-purpose architecture schemes, respectively. Compared to the all 2-qubit bus and the eff-5-freq's architecture schemes, after qubit mapping, the architecture schemes of our method have the smallest average growth of gate operations in multiple quantum programs (the largest average growth is 5.63%), which further supports the execution of different quantum programs. Meanwhile, the architecture schemes of our method also reduce the probability of frequency collisions by at least 4.48% compared to all other schemes. Furthermore, we compared our method with another special-purpose design method. In the schemes of different special-purpose architecture design methods, our method is able to generate architectures with better matching for multiple quantum programs. Therefore, our method can provide superconducting quantum processor architecture design with higher reusability for multiple quantum programs.

Keywords Superconducting quantum processor · Architecture design · Reusability improvement · Frequency collisions · Maximum common edge subgraph · Optimization problem

1 Introduction

Quantum computing is a new paradigm that takes advantage of quantum mechanical principles for computation. It has been proven to have magnificent acceleration effects than classical computing for solving computational problems in the fields of cryptography [1], quantum simulation [2], chemistry [3], and machine learning [4]. Currently, there are various schemes for implementing physical quantum computers, including superconducting qubits [5], trapped ions [6], silicon [7], and photons [8]. Among them, due to the advantages of superconducting qubits in high designability, scalability and controllability [9, 10], superconducting quantum systems have become a leading candidate in the development of quantum computers [11].

To make superconducting quantum systems more efficient, an important task is to improve the performance of superconducting quantum processors. Currently, researchers are not only making great efforts to improve metrics such as the qubit number [12–15], coherence time [16, 17], and gate fidelity [18–21], but also paying more attention to the optimization of processor architecture, such as optimizing the layout and connectivity between qubits on the processsor. The increase in connectivity between qubits can enable more physical qubit pairs to directly support two-qubit gates which enable to reduce the overhead of qubit mapping and routing, and optimize program execution [22]. In Ref. [23], experimental comparisons were made between quantum computers and corresponding processor architectures implemented in superconducting qubit and trapped ion systems. The authors indicate that the presence of more qubit connections in the processor architecture can help run quantum programs with more complex interactions. Refs. [24, 25] further compared more gen-

eral quantum processor architectures and emphasized the importance of having more connections between qubits. IBM proposed a max 4-qubit bus scheme based on the general-purpose square lattice architecture (IBM Q20 Tokyo [26]), which can better meet the operational requirements of different quantum programs by adding more qubit connections. However, in the scheme based on fixed-frequency superconducting qubits, more connections would increase the probability of frequency collisions between qubits, which could result in the increase in gate errors [27]. Therefore, IBM further proposed the heavy hex lattice architecture to reduce frequency collisions [28], but this scheme reduces the connectivity between qubits, which may slow down the execution speed of quantum programs.

To run programs more efficiently and reduce the impact of other physical factors, scholars propose to combine processor architecture design with quantum program. That is, designing special-purpose quantum processor to optimize program execution (minimizing gate operations after qubit mapping). Refs. [29, 30] proposed specific quantum processor architectures for Fermi Hubbard model simulation and computational chemistry problems to optimize program execution. Refs. [31, 32] customized the qubit connections on the processor architecture based on the given quantum program, resulting in a reduction of mapping overhead and an improvement of circuit fidelity, respectively. Refs. [33, 34] proposed two processor architecture design methods to trade-off the performance and frequency collisions of processor based on fixed-frequency superconducting qubit scheme. They used a heuristic workflow and an improved genetic algorithm combined with graph theory to design special-purpose processor architectures for quantum programs, respectively. Compared to the general-purpose design architecture schemes, they have better simulation results.

Although current research can provide processor architecture schemes that speed up execution for the given quantum program, these solutions lack support for the post-mapping gate operations' optimization of multiple programs within a single architecture. To improve the execution efficiency of different quantum programs, processor architectures need to be designed separately. That is to say, the reusability of current special-purpose processors is a critical problem to be improved. In addition, considering the limitations of the fabrication cost of multiple quantum processors, the low temperature measurement and multiple quantum processor interconnection technology (although there have been breakthroughs [35–38]), achieving efficient execution for different quantum programs by interconnecting multiple special-purpose quantum processors still faces enormous challenges. Therefore, one possible solution could be designing the processor architecture with higher compatibility, so that one single processor can be efficiently applied to multiple quantum programs. That is, one can improve the reusability of a quantum processor to support the execution optimization of different quantum programs.

In this paper, we focus on fixed-frequency superconducting quantum processors and propose a processor architecture design method for improving the reusability of special-purpose superconducting quantum processor based on graph theory. Our method constructs the processor architecture by finding the Maximum Common Edge Subgraph (MCES) [39] among multiple special-purpose processor architectures, in order to support the optimization of the post-mapping gate count in multiple quantum programs, reduce the probability of frequency collisions and improve the reusabil-

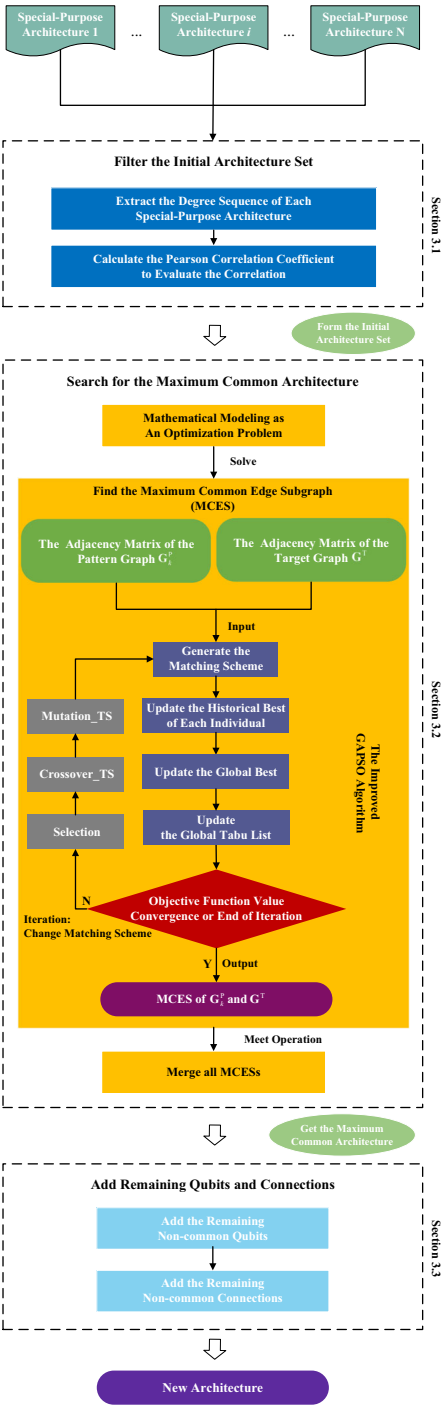

Fig. 1 Overview of basic ideas of the method

ity of special-purpose processors. The specific flow is shown in Fig. 1. Firstly, the initial architecture set is filtered by evaluating the correlation of degree sequences (calculating the Pearson correlation coefficient [40]) among multiple special-purpose processor architectures; then, the maximum common structure is obtained in the set by searching for the MCES of different pattern graphs and the target graph (Here, we model the process of searching for MCES as an optimization problem and improve the GAPSO algorithm [41] to solve it.); finally, non-common remaining qubits and connections are added to the maximum common substructure to generate a novel processor architecture. To verify the effectiveness of our method, we selected quantum programs that cover different functions under 9 types of qubit numbers, and conducted simulation experiments on the architecture scheme of our method, the architecture scheme designed by the eff-5-freq algorithm in Ref. [33], and two general-purpose architectures based on square-lattice (all 2-qubit bus and max 4-qubit bus). By jointly comparing the average growth of the total post-mapping gate count of different architectures on multiple quantum programs and their impact on frequency collisions, we find that the architecture scheme generated by our method is superior to the three compared ones. Compared to the all 2-qubit bus and the eff-5-freq's architecture schemes, after mapping multiple quantum programs, the average growth in gate operations of ours is the smallest, with a maximum average increase of 5.63%. Here, the comparisons with the max 4-qubit bus architecture are not included because it has a higher probability of frequency collisions. Moreover, among all the compared architectures, the architecture schemes of our method further effectively reduce the probability of frequency collisions, with a minimum reduction of 4.48%. This demonstrates the effectiveness of our method in improving the reusability of special-purpose superconducting quantum processors. In addition, we compared our method with another special-purpose architecture design method proposed in Ref. [34]. In the architecture schemes designed by different special-purpose architecture design methods, our method can also generate architectures with better matching for multiple quantum programs. This also demonstrates that our method can be adapted to different special-purpose architecture design methods, thus proving the reliability of our method.

The rest of this paper is organized as follows. In Sect. 2, we review the necessary basics related to our research. In Sect. 3, we describe the three steps of the superconducting quantum processor architecture design method in detail. In Sect. 4, we make the detailed comparisons of the optimization results between different architectures on the total post-mapping gate count and frequency collisions. Finally, we summarize our work and discuss directions for future research.

2 Preliminaries

In this section, we will briefly review the necessary basics related to our research.

2.1 Pearson correlation coefficient

The Pearson correlation coefficient r is a statistical measure used to quantify the correlation between two variables, $\boldsymbol{X}$ and $\boldsymbol{Y}$, with values ranging from -1 to 1 [40].

The formula is given by Eq. (1). When r is 1, it means that the two variables have a perfect positive correlation, indicating that their variation trends are completely consistent. When r is -1, it means that the two variables have a perfect negative correlation, indicating that their variation trends are completely opposite. When r is 0, it means that there is no correlation between the two variables.

$$r = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum {x_i}^2 - \left(\sum x_i\right)^2}\sqrt{n\sum {y_i}^2 - \left(\sum y_i\right)^2}} \tag{1}$$

2.2 Maximum common edge subgraph (MCES)

The maximum common edge subgraph [39] problem is described as follows:

Given: two connected graphs **G** and **H**, both of which have the same number of vertices $|\mathrm{V_G}| = |\mathrm{V_H}|$ and non-empty sets of edges $|\mathrm{E_G}| \neq \phi$, $|\mathrm{E_H}| \neq \phi$.

Find: a common subgraph with the maximum number of edges of **G** and **H** and not necessarily an induced subgraph.

If **G** and **H** are defined as the architecture graph of two different special-purpose processors, the MCES of both retains the most connections in each special-purpose processor architecture. In qubit mapping, when different programs are run on the architecture containing MCES, matched connections can reduce additional gate operations (such as SWAP gates), improving the execution efficiency of different programs. Therefore, to reduce additional gate operations for multiple quantum programs, one should construct the new processor architecture based on the MCES among multiple special-purpose processor architectures. According to the definition of MCES, multiple special-purpose architectures should maintain a consistent number of qubits.

2.3 GAPSO algorithm

GAPSO is a heuristic algorithm that combines genetic algorithm [42] and particle swarm optimization [43]. It enhances the global search capability of particle swarm optimization by using the crossover and mutation operations of genetic algorithm, and guides the genetic process through the individual update rules of particle swarm optimization. GAPSO algorithm combines the advantages of genetic algorithm and particle swarm optimization, which can effectively solve multi-dimensional, nonlinear, and complex optimization problems with fast convergence speed and high optimization accuracy.

2.4 Frequency collisions and yield rate

Due to the limitations of the fabrication process, there will be a certain deviation between the actual value and the designed value of qubit frequency when fabricating superconducting quantum processor. Assuming the designed frequency of a physical qubit is f, the actual frequency after fabrication will be $f + N(0,\sigma_f)$. $N(0,\sigma_f)$ is a Gaussian noise with σ_f being the fabrication precision parameter (standard deviation)

[27, 33]. Due to the deviation of σ_f, when two or three qubits are connected, their frequencies may satisfy some specific conditions (Referring to Figure 4 in Ref. [22] for more details.), and meeting these conditions means that frequency collisions occur. The 'yield rate' is used to estimate the probability of not having frequency collisions [27]. For a completed design of the superconducting quantum processor, we use Monte Carlo simulation to estimate its yield rate [27, 33]. The higher the yield rate value, the lower the probability of frequency collisions occurring in the superconducting quantum processor. Here, our qubit frequency design follows the IBM's 5-frequency scheme [27]. The detailed calculation method for yield rate can be found in Refs. [33, 34].

3 Method

In this section, we will describe the three steps of the processor architecture design method in detail.

3.1 Filtering the initial architecture set

To support the optimization of the post-mapping gate operations in different programs, the processor architecture should retain as many connections as possible from each special-purpose processor architecture. Therefore, the selected special-purpose processor architectures should have a higher degree of similarity to ensure that the MCES contains more common edges. Here, we calculate the Pearson correlation coefficient based on the degree sequences corresponding to different architecture graphs, in order to select special-purpose architectures that may have more common edges as the initial architecture set. We take the calculation of Pearson correlation coefficient between two different special-purpose processor architectures (with the same number of qubits) as an example to illustrate the specific process. First, extract the degree sequences (non-increasing sequences) of two architecture graphs ($\mathbf{G}_1$ and $\mathbf{G}_2$). $\boldsymbol{X}$ and $\boldsymbol{Y}$ are the degree sequences of $\mathbf{G}_1$ and $\mathbf{G}_2$, respectively. $\boldsymbol{X} = [x_1, x_2, \ldots, x_i, \ldots, x_n]$, $x_1 \geq x_2 \geq \cdots \geq x_i \geq \cdots \geq x_n$, with x_i being the degree of one vertex on $\mathbf{G}_1$, $\boldsymbol{Y} = [y_1, y_2, \ldots, y_i, \ldots, y_n]$, $y_1 \geq y_2 \geq \cdots \geq y_i \geq \cdots \geq y_n$, with y_i being the degree of one vertex on $\mathbf{G}_2$, and n is the number of vertices; then, bring $\boldsymbol{X}$, $\boldsymbol{Y}$ into Eq.(1) to calculate the Pearson correlation coefficient r; finally, determine whether $|r|$ has reached the threshold.

According to statistical knowledge, when $|r| \geq 0.8$, it is considered that the calculated two degree sequences have a strong correlation [40]. Although the strong correlation of degree sequences is not a sufficient and necessary condition to determine whether two graphs have great similarity, when there is greater similarity between two architecture graphs, the value of $|r|$ is usually greater. Conversely, when the value of $|r|$ is greater, the probability of great similarity between the two architecture graphs is higher. Therefore, when $|r| \geq 0.8$, we consider that the two architectures may have a sufficiently large number of common edges, and include both in the initial architecture set. Subsequently, the MCES of both architectures is found according to the work-

flow in Sect. 3.2. When there are multiple different special-purpose architectures, we choose the architecture with the largest number of edges as the target graph $\mathbf{G}^{\mathrm{T}}$, and each of the remaining architectures as the pattern graph $\mathbf{G}_k^{\mathrm{P}}$ (k is the index of the pattern graph), respectively. Choosing the special-purpose architecture with the largest number of edges as $\mathbf{G}^{\mathrm{T}}$ allows the pattern graph to match more common edges in solving MCES, as it contains more qubit connections. In this way, more connections among different special-purpose architectures can be retained in the new processor architecture constructed based on MCES, to reduce qubit mapping overhead and improve the reusability of the architecture. Then, calculate the values of r for $\mathbf{G}^{\mathrm{T}}$ and each $\mathbf{G}_k^{\mathrm{P}}$ in turn. Finally, $\mathbf{G}_k^{\mathrm{P}}$ with $|r| \geq 0.8$ is selected and together with $\mathbf{G}^{\mathrm{T}}$ form the initial architecture set.

3.2 Searching for the maximum common structure

Based on the architecture set selected in Sect. 3.1, this section explains how to determine the maximum common structure in the architecture set by finding the MCESs among the target graph and different pattern graphs. The MCES problem has been proven to be a NP-hard problem [39]. Here, we construct the mathematical model for solving MCES as an optimization problem, and improve the GAPSO algorithm (heuristic algorithm) to obtain the approximate optimal solution. The details are as follows.

3.2.1 Mathematical model

Assume that the initial architecture set contains N special-purpose processor architectures $\mathbf{G}_l(\mathrm{V}_l,\mathrm{E}_l)$, $1 \leq l \leq N$, and $|\mathrm{V}_1| = |\mathrm{V}_2| = \cdots = |\mathrm{V}_l| = \cdots = |\mathrm{V}_N| = n$ (i.e., each architecture contains n qubits). Similarly, following the definition in Sect. 3.1, we select the architecture with the most edges as the target graph $\mathbf{G}^{\mathrm{T}}(\mathrm{V}^{\mathrm{T}}, \mathrm{E}^{\mathrm{T}}) = \mathbf{G}_l$ with $\max |\mathrm{E}_l|$, and the remaining N-1 architectures are pattern graphs $\mathbf{G}_k^{\mathrm{P}}(\mathrm{V}_k^{\mathrm{P}}, \mathrm{E}_k^{\mathrm{P}}, \mathrm{M}_k)$. Here, M_k is the matching scheme for $\mathrm{V}_k^{\mathrm{P}}$ to V^{T}. According to M_k, one can determine the number of common edges of $\mathbf{G}_k^{\mathrm{P}}$ and $\mathbf{G}^{\mathrm{T}}$ by adjusting the adjacency matrix $\mathbf{A}_k^{\mathrm{P}}$ of $\mathbf{G}_k^{\mathrm{P}}$. Note that the adjusted adjacency matrix of $\mathbf{G}_k^{\mathrm{P}}$ is $\mathbf{A}_k^{\mathrm{P}'}$. By converting $\mathrm{V}_k^{\mathrm{P}}$ to V^{T} according to the matching scheme of M_k, the edge set of $\mathbf{G}_k^{\mathrm{P}}$ can be converted to the edge set corresponding to the serial number of vertices in $\mathbf{G}^{\mathrm{T}}$. Equation(2) is the edge set conversion formula.

$$\begin{aligned} &\mathbf{A}_k^{\mathrm{P}'}[\mathrm{M}_k(i)][\mathrm{M}_k(j)] = \mathbf{A}_k^{\mathrm{P}}[i][j] \\ &1 \leq k \leq N \wedge k \neq l \wedge \mathrm{M}_k(i), \mathrm{M}_k(j) \in \mathrm{V}^{\mathrm{T}} \end{aligned} \tag{2}$$

Since the adjacency matrix takes the value 0 or 1, the meet operation ($\wedge$) of the boolean matrix can be used. Then the number of common edges between $\mathbf{G}_k^{\mathrm{P}}$ and $\mathbf{G}^{\mathrm{T}}$ through $\mathbf{A}_k^{\mathrm{P}'} \wedge \mathbf{A}^{\mathrm{T}}$ ($\mathbf{A}^{\mathrm{T}}$ is the adjacency matrix of $\mathbf{G}^{\mathrm{T}}$) can be obtained with Eq.(3).

$$\sum_{i=0}^{n-1}\sum_{j=i+1}^{n-1}(\mathbf{A}_k^{\mathrm{P}'} \wedge \mathbf{A}^{\mathrm{T}})[i][j] \tag{3}$$

If the architecture of $\mathbf{A}_k^{\mathrm{P}'} \wedge \mathbf{A}^{\mathrm{T}}$ is a connected graph, a common subgraph of $\mathbf{G}_k^{\mathrm{P}}$ and $\mathbf{G}^{\mathrm{T}}$ is found. Finding the MCES of $\mathbf{G}_k^{\mathrm{P}}$ and $\mathbf{G}^{\mathrm{T}}$ to maximize the common edge set requires searching for an optimal matching scheme M_k. Therefore, this process can be constructed as a mathematical model of the optimization problem, *i.e.*,, Eq.(4). $f(\mathrm{M}_k)$ is the objective function. When $\mathbf{A}_k^{\mathrm{P}'} \wedge \mathbf{A}^{\mathrm{T}}$ is a connected graph, the function value is the number of common edges. Otherwise, the value is -10. Here, we exclude the case where $\mathbf{A}_k^{\mathrm{P}'} \wedge \mathbf{A}^{\mathrm{T}}$ is not a connected graph. The reason is that there may be multiple situations when make $\mathbf{A}_k^{\mathrm{P}'} \wedge \mathbf{A}^{\mathrm{T}}$ connected (there may be adding different edges to make it connected), which increases the complexity of processor architecture design. By constantly searching, the MCES $\mathbf{G}_k^{\mathrm{M}}$ of $\mathbf{G}_k^{\mathrm{P}}$ and $\mathbf{G}^{\mathrm{T}}$ can be obtained by maximizing the function value of $f(\mathrm{M}_k)$ while ensuring that the architecture of $\mathbf{A}_k^{\mathrm{P}'} \wedge \mathbf{A}^{\mathrm{T}}$ is a connected graph. Figure 2 is an example for finding the MCES of $\mathbf{G}_k^{\mathrm{P}}$ and $\mathbf{G}^{\mathrm{T}}$, where $\mathbf{A}_k^{\mathrm{M}}$ is the adjacency matrix of the MCES. Among them, the connections between qubits adopt the 2-qubit bus, 3-qubit bus, and 4-qubit bus introduced in Ref. [33].

3.2.2 Solving the optimization problem

Based on the mathematical model constructed in Sect. 3.2.1, we improved the GAPSO algorithm to solve the problem. The objective of the improvement is to accelerate the solution of the optimal matching solution M_k and the MCES $\mathbf{G}_k^{\mathrm{M}}$ by reducing the repeated times of the search process. Below, we will provide a detailed description of how to improve the GAPSO algorithm to solve the optimization problem.

$$\max f(\mathrm{M}_k)$$
$$f(\mathrm{M}_k) = \begin{cases} \sum_{i=0}^{n-1}\sum_{j=i+1}^{n-1}(\mathbf{A}_k^{\mathrm{P}'} \wedge \mathbf{A}^{\mathrm{T}})[i][j], & \mathbf{A}_k^{\mathrm{P}'} \wedge \mathbf{A}^{\mathrm{T}} \text{ is connected graph} \\ -10, & \text{otherwise} \end{cases} \tag{4}$$

Firstly, we determine the encoding method for individuals. Take each matching scheme M_k as an individual, where M_k is an n-dimensional vector and n is the number of qubits. As defined in the previous section, $\mathrm{M}_k(i) \in \mathrm{V}^{\mathrm{T}}, i \in \mathrm{V}_k^{\mathrm{P}}, 0 \leq i, \mathrm{M}_k(i) \leq n-1$, the scale of the search space is $n!$. Taking $f(\mathrm{M}_k)$ as the fitness function, the optimal individual corresponds to MCES, $\mathbf{G}_k^{\mathrm{M}}$.

The main improvement is to introduce the idea of Tabu Search (TS) into the GAPSO algorithm to increase the search efficiency. TS is a heuristic optimization algorithm used to find the optimal or suboptimal solution in a large-scale search space. In TS, a tabu list is constructed to record the solutions that have been visited during the search process, in order to avoid revisiting some solutions that have already been searched and prevent getting trapped in local optima during the search process. The specific improvements are as follows:

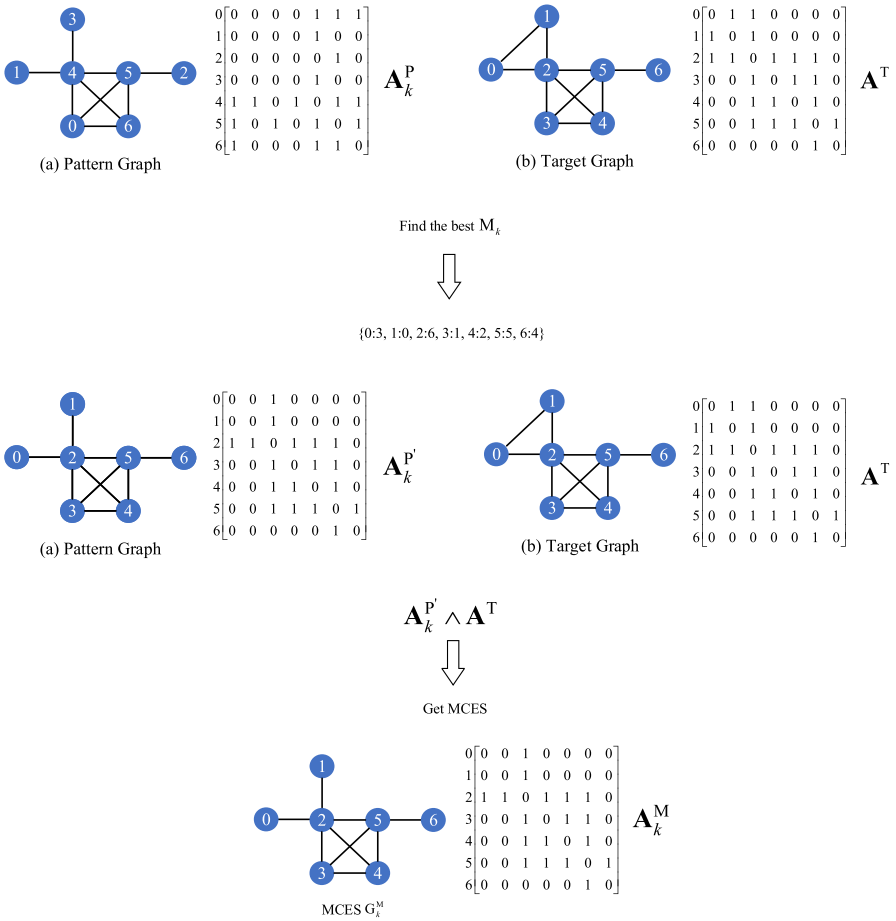

Fig. 2 Example of MCES construction for $\mathbf{G}_k^P$ and $\mathbf{G}^T$

1. Construct a global tabu list to store the best individual of each iteration as well as serving as the tabu object for all crossover and mutation operations;
2. Construct local tabu tables for crossover and mutation operations, respectively. For crossover operations, the stored tabu objects are the parent population of the previous generation and the offspring population generated by crossover operation. The corresponding pseudo code is shown in Algorithm 1. For mutation operations, the stored tabu objects are the parent population of the previous generation, the population after crossover, and the offspring population generated by mutation operation. The corresponding pseudo code is shown in Algorithm 2.

By constructing global and local tabu lists, the GAPSO algorithm avoids repeating searches in the search space, which improves the search efficiency. Moreover, according to the searched tabu objects, the algorithm can effectively prevent the algorithm from falling into the local optimal solution. The overall pseudo code of the improved GAPSO algorithm is shown in Algorithm 3.

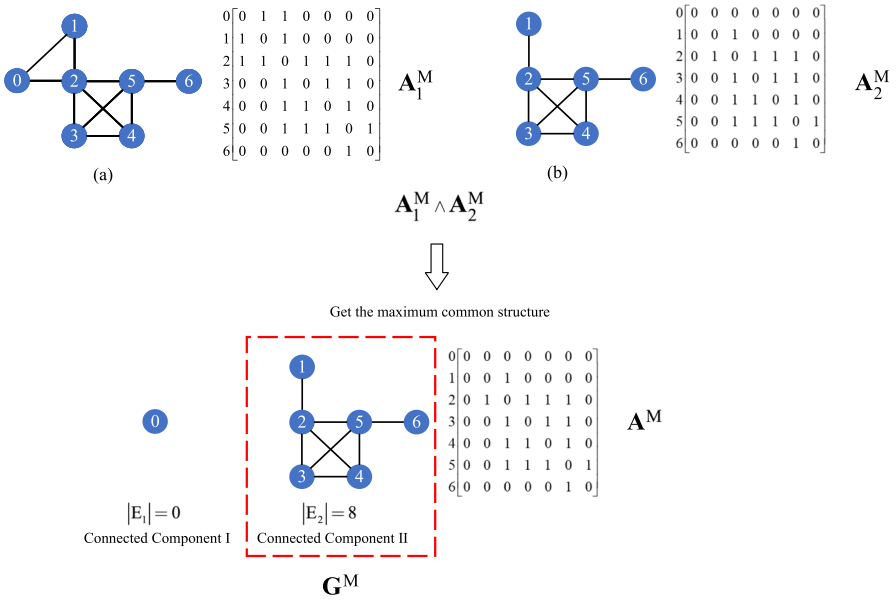

Fig. 3 Example of the maximum common structure $\mathbf{G}^{\mathrm{M}}$

When N-1 MCESs are found, their adjacency matrices are subjected to the meet operation $\mathbf{A}_1^{\mathrm{M}} \wedge \mathbf{A}_2^{\mathrm{M}} \wedge \cdots \wedge \mathbf{A}_k^{\mathrm{M}}$, $1 \leq k \leq N \wedge k \neq l$, and the result is recorded as $\mathbf{A}^{\mathrm{M}}$. If $\mathbf{A}^{\mathrm{M}}$ can be represented by a connected graph, then it is represented by the maximum common structure $\mathbf{G}^{\mathrm{M}}$. Otherwise, the connected component with the maximum number of edges is represented by $\mathbf{G}^{\mathrm{M}}$. For example, Fig. 3a, b represents two MCESs, respectively. $\mathbf{A}^{\mathrm{M}}$ is the result of the meet operation ($\wedge$) performed for two MCESs' adjacency matrices. It can be seen that $\mathbf{A}^{\mathrm{M}}$ contains two connected components. We choose the connected component with the maximum number of edges (Connected Component II) as $\mathbf{G}^{\mathrm{M}}$.

3.3 Adding remaining qubits and connections

After obtaining the maximum common structure $\mathbf{G}^{\mathrm{M}}$, there may be cases where some qubits and their corresponding connections are missing. Figure 4 is a specific example. Figure 4a, b are two special-purpose architectures (6 Qubits), and $\mathbf{G}^{\mathrm{M}}$ is the maximum common structure between them. It can be seen that one qubit and the corresponding connection are missing in $\mathbf{G}^{\mathrm{M}}$. To meet the running requirements of the original programs on architectures (a) and (b) in Fig. 4, the remaining non-common qubits and connections need to be added to complete the final architecture design. Here, we design the adding rules for the purpose of reducing the probability of frequency collisions. The specific rules are as follows:

1. Choose vertices in $\mathbf{G}^{\mathrm{M}}$ with degrees no larger than 2 as insertion positions.

Algorithm 1 Crossover_TS algorithm

Input: global tabu list tabu_list_global, parent population parents_pop, historical best of each individual p_best, global best individual g_best, inertia weight w, cognitive factor c_1, social factor c_2
Output: Crossed population children_pop

```
Initialization: children_pop ← ϕ
for i←1 to len(parents_pop) do
    flag = True
    while flag do
        parent_1 = parents_pop[i]
        if random ≤ w then
            // random takes value less than or equal to w+c1+c2
            parent_2 = change(parents_pop[i]) // Random pairs exchange
        else if random ≤ w+c1 then
            parent_2 = p_best[i]
        else
            parent_2 = g_best
        end if
        if meet crossover condition then
            child = PBX_crossover(parent_1, parent_2)
            // Use Position-based crossover and choose the one with higher fitness value between the two crossovered individuals
        else
            child = change(parents_pop[i])
        end if
        if (child not in tabu_list_global) & (child not in children_pop) & (child not in parents_pop) then
            // Global tabu and local tabu judgment, local tabu list includes children_pop and parents_pop
            children_pop.append(child)
            flag = False
        end if
    end while
end for
Return children_pop
```

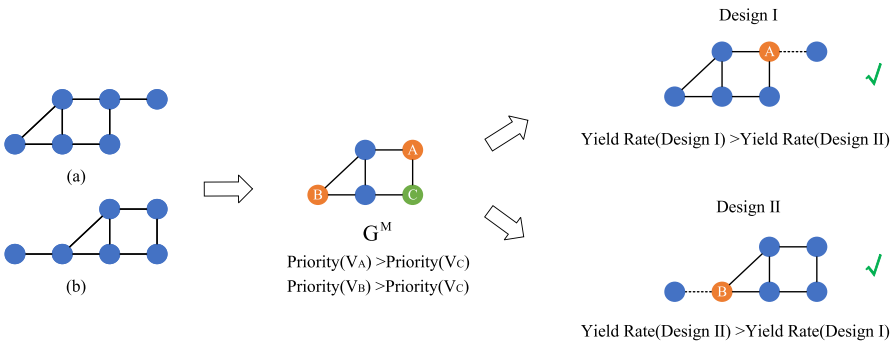

Fig. 4 Example of adding remaining qubits and connections

2. Among the positions selected in rule 1, prioritize selecting the positions that can restore the original architecture by adding remaining qubits and connections.

Rule 1 reduces the probability of frequency collisions by decreasing the connectivity of qubits. Rule 2 prioritizes the restoration of the original special-purpose architecture as much as possible, while ensuring a reduction in the probability of frequency col-

Algorithm 2 Mutation_TS algorithm

Input: global tabu list tabu_list_global, parent population parents_pop, population to be mutated after crossover children_pop_not_mutate
Output: Mutated population children_pop
Initialization: children_pop ← ϕ
for i←1 to len(children_pop_not_mutate) **do**
  flag = True
  while flag **do**
    if meet mutation condition **then**
      child_mutate =
mutate(children_pop_not_mutate[i]) // Random multiple pair exchange
    else
      child_mutate = children_pop_not_mutate[i]
    end if
    if (child_mutate not in tabu_list_global) & (child_mutate not in children_pop_not_mutate) & (child_mutate not in children_pop) & (child_mutate not in parents_pop) **then**
      // Global tabu and local tabu judgment, local tabu list includes children_pop, parents_pop and children_pop_not_mutate
      children_pop.append(child_mutate)
      flag = False
    end if
  end while
end for
Return children_pop

lisions, in order to decrease the overhead of qubit mapping. According to our rules, three optional insertion positions (A, B and C) are marked on $\mathbf{G}^{\mathrm{M}}$ in Fig. 4. The orange vertices (A and B) indicate that after adding qubit and connection, they can maintain consistency with the original architectures, while the green vertex C represents the new insertion position. According to Rule 2, the orange vertices have a higher priority than the green vertex. Moreover, if there are multiple choices for insertion positions of different categories, we prioritize selecting the architecture solution with the lowest probability of frequency collisions after adding qubits and connections. Here, we add remaining qubit and connection to the orange vertices (A and B) separately, calculate the yield rate values for the two designs (according to Sect. 2.4, the higher the yield rate, the lower the probability of frequency collisions), and finally choose the architecture scheme with a higher yield rate value.

4 Simulation

To verify the effectiveness of our method, we selected quantum programs with different functions and qubit numbers (from 8 to 16), and conducted simulation experiments on the architecture of our method, the architecture designed by the eff-5-freq algorithm [33], and two general-purpose architectures based on the square lattice. Here, all the quantum programs were selected from ibm_qx_mapping benchmark [44], and 3 quantum programs were selected according to the rules in Sect. 3.1 for each number of qubits. The eff-5-freq algorithm generates special-purpose architectures for each quantum program. Our method takes the three special-purpose architectures generated

Algorithm 3 Improved GAPSO algorithm

Input: qubit number q_num, adjacency matrix $\mathbf{A}_k^{\mathrm{P}}$ of $\mathbf{G}_k^{\mathrm{P}}$, adjacency matrix $\mathbf{A}^{\mathrm{T}}$ of $\mathbf{G}^{\mathrm{T}}$, population size pop_size, iterations iter_num, global tabu list length limit_length, inertia weight w, cognitive factor c_1, social factor c_2
Output: $\mathbf{G}_k^{\mathrm{M}}$

```
Initialization: current_pop ← initial_population(q_num, pop_size) // Initialize population
p_best = current_pop // Initialize the historical best of each individual
g_best = p_best[max(fitness(p_best, A^T, A_k^P)).index] // Initialize the global best individual
tabu_list_global.append(g_best) // Initialize the global tabu list
for i←1 to iter_num do
    children_pop = Crossover_TS(tabu_list_global, current_pop, p_best, g_best, w, c1, c2)
    children_pop = Mutation_TS(children_pop, tabu_list_global, current_pop)
    for j←1 to pop_size do
        if fitness(children_pop[j], A^T, A_k^P) >fitness(current_pop[j], A^T, A_k^P) then
            // Selection
            current_pop[j] = children_pop[j]
        end if
        if fitness(current_pop[j], A^T, A_k^P) >fitness(p_best [j], A^T, A_k^P) then
            p_best[j] = current_pop[j] // Update p_best
        end if
    end for
    g_best = p_best[max(fitness(p_best, A^T, A_k^P)).index] // Update g_best
    if len(tabu_list_global) == limit_length then
        remove(tabu_list_global[0])
    end if
    tabu_list_global.append(g_best) // Update the global tabu list
end for
G_k^M = MCES(g_best, A^T, A_k^P) // Construct MCESs of G_k^P and G^T
Return G_k^M
```

by the eff-5-freq algorithm as input to design reusable architectures for each number of qubits. The two general-purpose architectures are all 2-qubit bus scheme and max 4-qubit bus scheme, respectively, as shown in Fig. 5. When the number of qubits is less than or equal to 9, the architecture comparison uses Fig. 5a, c. When the number of qubits is greater than 9 but less than or equal to 16, Fig. 5b, d are used for comparison. Note that, we did not select the max 4-qubit bus 4×4 scheme as it has a higher probability of frequency collisions. Instead, we chose max 4-qubit bus 2×8 scheme.

To evaluate the effect of the architectures in optimizing post-mapping gate operations, we compared the total post-mapping gate count of three quantum programs under each number of qubits on different architectures. A smaller value of the total post-mapping gate count indicates a better execution effect of the architecture (extra gate operations have been reduced) [33, 45, 46]. The qubit mapping algorithm we use is the SABRE algorithm [46]. To compare the impact of different architectures on frequency collisions, we selected yield rate for evaluation. The yield rate is used to estimate the probability of no frequency collisions occurring after frequency allocation [27, 47, 48]. The higher the yield rate value, the lower the probability of frequency collisions occurring. Detailed frequency allocation algorithm, yield rate simulation method, and parameter settings are referred to Refs. [30, 33]. The eff-5-freq algorithm may output multiple architecture schemes based on different quantum programs. Here, we choose the architecture with the best total post-mapping gate count and yield rate.

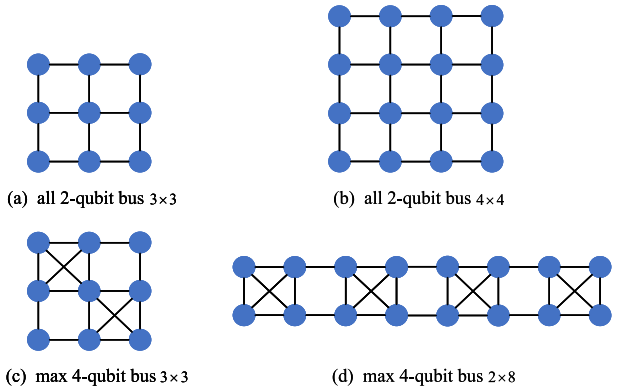

Fig. 5 General-purpose architectures for all 2-qubit bus and max 4-qubit bus schemes

Table 1 Algorithm parameter settings

Parameter	Value
Population size	300
Iterations	150
Inertia weight	0.2
Cognitive factor	0.4
Social factor	0.4
Crossover probability	0.85
Mutation probability	0.15

4.1 Parameter setting

The parameter settings are listed in Table 1. Since the scale of the search space is $n!$, we set the length of the tabu list to n^2. In addition, the equipment parameters for our experiment are: CPU Intel(R) Core(TM) i7-10510U, RAM 16 GB.

4.2 Result comparison

4.2.1 The improvement of GAPSO algorithm

To demonstrate the improvement effect, the improved GAPSO algorithm with TS idea (Sect. 3.2.2) is compared with the standard GAPSO algorithm. Figure 6 shows the process of solving MCES using two algorithms on 9-qubit and 16-qubit architectures. Here, we do not limit the iteration number. Moreover, when the fitness value of the algorithm does not change after 100 consecutive iterations, we record the time and the solution (the standard GAPSO algorithm is the black triangle and the improved GAPSO algorithm is red circle).

It can be seen that under 150 iterations, the improved GAPSO algorithm has higher fitness values (the number of common edges) than the standard GAPSO algorithm,

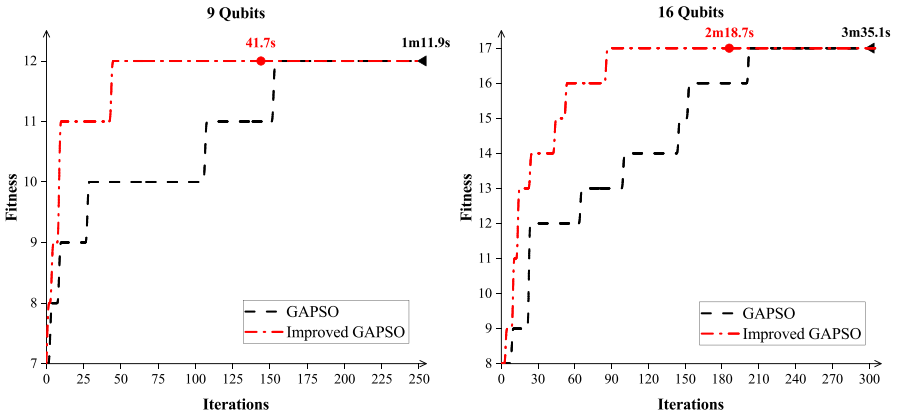

Fig. 6 Comparison of the standard GAPSO and the improved GAPSO

with the maximum common edge numbers increased by 1 and 2, respectively. When the iteration number limit is removed, both algorithms eventually converge to the same fitness values, but the improved GAPSO algorithm takes less time. The improved GAPSO algorithm reduced 30.2 s and 76.4 s, respectively. This indicates that the MCES solved by the improved GAPSO algorithm is better than that of the standard GAPSO algorithm and the improvement has enhanced the search efficiency of the algorithm. Moreover, by comparing the search processes of the two algorithms, the improved GAPSO algorithm that introduces tabu list can jump out of local optima and find better MCES during the solving process. Thus, the effectiveness of the improvement strategy in our method is verified.

4.2.2 Comparison of execution effects on different architectures

The total gate count of each quantum program after mapping (total post-mapping gate count) on the corresponding special-purpose architecture (generated by the eff-5-freq algorithm) is taken as the baseline value. And we compare the changes in the total post-mapping gate count relative to the baseline value after mapping the quantum programs onto other architectures, as shown in Table 2. Table 2 contains 9 subtables, each of which has the same number of qubits for three quantum programs.

In each subtable, it can be seen that compared to the other architecture schemes (non-corresponding special-purpose architectures, all 2-qubit bus and ours), quantum programs have smaller total post-mapping gate counts both on the corresponding special-purpose architecture schemes (generated by the eff-5 freq algorithm) and the max 4-qubit bus architecture scheme. Moreover, under the max 4-qubit bus architecture scheme, the total post-mapping gate count of each program is mostly lower than the baseline value, and the maximum reduction is 17.44% (Table 2(b)). This is because the max 4-qubit bus scheme has more qubit connections and reduces more qubit mapping overhead compared to other schemes. When different programs are run on the remaining architectures, the architecture schemes of our method are better in terms of total post-mapping gate count comparison. In the last row of each sub-

Table 2 Comparison of execution effects between different architectures in each qubit numbers

(a) 8 Qubits	eff-5-freq			(a)	(c)	Our Method
	rd53_251	urf2_277	f2_232			
rd53_251	0%	4.51%	5.64%	9.40%	− 14.45%	**1.07%**
urf2_277	6.59%	0%	6.41%	8.96%	− 8.85%	**6.25%**
f2_232	14.60%	6.01%	0%	16.93%	− 4.11%	**2.39%**
$\|r\|$	0.8090		0.9045			
Avg	7.06%	3.51%	4.02%	11.76%		**3.24%**
(b) 9 Qubits	eff-5-freq			(a)	(c)	Our Method
	con1_216	hwb8_113	urf5_280			
con1_216	0%	3.86%	1.89%	6.52%	− 9.68%	**0.77%**
hwb8_113	4.14%	0%	3.28%	3.45%	− 13.07%	**2.71%**
urf5_280	3.02%	1.70%	0%	2.99%	− 17.44%	**1.31%**
$\|r\|$	0.8513		0.9364			
Avg	2.39%	1.85%	1.72%	4.32%		**1.60%**
(c) 10 Qubits	eff-5-freq			(b)	(d)	Our Method
	mini_alu_305	sys6-v0_111	rd73_140			
mini_alu_305	0%	1.87%	10.49%	5.99%	− 10.86%	**0%**
sys6-v0_111	2.36%	0%	7.40%	17.85%	− 1.68%	**0.67%**

Table 2 continued

(c) 10 Qubits	eff-5-freq			(b)	(d)	Our Method
	mini_alu_305	sys6-v0_111	rd73_140			
rd73_140	11.22%	9.62%	0%	19.55%	7.05%	**6.73%**
$\|r\|$		0.9494	0.8750			
Avg	4.53%	3.83%	5.96%	14.46%		**2.47%**
(d) 11 Qubits	eff-5-freq			(b)	(d)	Our Method
	wim_266	dc1_220	z4_268			
wim_266	0%	9.79%	10.42%	18.40%	3.52%	**3.21%**
dc1_220	20.82%	0%	11.17%	26.07%	0.72%	**8.55%**
z4_268	9.41%	6.21%	0%	13.37%	0%	**3.77%**
$\|r\|$	0.9116		0.9629			
Avg	10.08%	5.33%	7.20%	19.28%		**5.18%**
(e) 12 Qubits	eff-5-freq			(b)	(d)	Our Method
	sym9_146	sqrt8_260	cycle10_2_110			
sym9_146	0%	1.66%	7.07%	12.06%	− 2.91%	**0.83%**
sqrt8_260	10.96%	0%	3.53%	10.98%	− 1.27%	**0.56%**
cycle10_2_110	8.12%	9.30%	0%	10.66%	− 3.61%	**7.69%**

Table 2 continued

(e) 12 Qubits	eff-5-freq			(b)	(d)	Our Method
	sym9_146	sqrt8_260	cycle10_2_110			
$\|r\|$	0.9322	0.9224				
Avg	6.36%	3.65%	3.53%	11.23%		**3.03%**

(f) 13 Qubits	eff-5-freq			(b)	(d)	Our Method
	rd53_311	adr4_197	root_255			
rd53_311	0%	12.24%	8.47%	11.76%	−11.53%	**5.41%**
adr4_197	11.27%	0%	11.35%	11.84%	0.24%	**7.07%**
root_255	3.96%	3.40%	0%	7.04%	−4.66%	**1.09%**
$\|r\|$	0.9260		0.9251			
Avg	5.08%	5.21%	6.61%	10.21%		**4.52%**

(g) 14 Qubits	eff-5-freq			(b)	(d)	Our Method
	0410184_169	sym6_316	sao2_257			
0410184_169	0%	13.62%	19.60%	27.57%	−9.97%	**10.30%**
sym6_316	15.78%	0%	5.80%	9.28%	−3.48%	**2.55%**
sao2_257	6.30%	5.83%	0%	6.95%	0%	**4.03%**

Table 2 continued

(g) 14 Qubits	eff-5-freq			(b)	(d)	Our Method
	0410184_169	sym6_316	sao2_257			
$\|r\|$	0.9323	0.9591				
Avg	7.36%	6.48%	8.47%	14.60%		**5.63%**

(h) 15 Qubits	eff-5-freq			(b)	(d)	Our Method
	misex1_241	ham15_107	co14_215			
misex1_241	0%	9.46%	10.30%	19.13%	4.70%	**4.72%**
ham15_107	9.28%	0%	7.32%	11.74%	2.24%	**4.77%**
co14_215	4.25%	4.59%	0%	5.71%	− 1.74%	**3.12%**
$\|r\|$	0.9333		0.9361			
Avg	4.51%	4.68%	5.87%	12.19%		**4.20%**

(i) 16 Qubits	eff-5-freq			(b)	(d)	Our Method
	inc_237	mlp4_245	cnt3-5_180			
inc_237	0%	7.79%	7.36%	15.02%	1.00%	**5.56%**
mlp4_245	3.59%	0%	3.88%	6.45%	− 1.78%	**2.41%**
cnt3-5_180	5.83%	5.42%	0%	10.69%	− 15.56%	**1.25%**
$\|r\|$	0.9311		0.9025			
Avg	3.14%	4.40%	3.75%	10.72%		**3.07%**

table, we compared the average growth in total post-mapping gate count after running each quantum program with different architectures. The max 4-qubit bus scheme is not compared here because the architecture is more prone to frequency collisions (see Sect. 4.2.3 for details). It can be seen that the architecture scheme of our method has the lowest average growth for the three quantum programs in different subtables, ranging from 1.6% to 5.63%. This is due to the fact that our method is based on the maximum common structure of multiple special-purpose architectures to construct a new architecture, which preserves more connections of each special-purpose architecture. When different quantum programs are run on our method's architecture, the matching connections can reduce the extra gate operations during qubit mapping.

In addition, to verify the rationality of using Pearson correlation coefficient, we have listed the values of $|r|$ calculated by the degree sequence of the target graph $\mathbf{G}^{\mathrm{T}}$ and pattern graph $\mathbf{G}_k^{\mathrm{P}}$ in each subtable. In each subtable, the special-purpose architecture with a non empty $|r|$ value represents the pattern graph, otherwise it is the target graph. By jointly comparing the changes in $|r|$ and total post-mapping gate count, it can be seen that the higher $|r|$ of the degree sequences of $\mathbf{G}_k^{\mathrm{P}}$ and $\mathbf{G}^{\mathrm{T}}$, the less gate operations added to the quantum program corresponding to $\mathbf{G}^{\mathrm{T}}$ on the $\mathbf{G}_k^{\mathrm{P}}$ architecture. This verifies the rationality of using Pearson correlation coefficient of degree sequence for filtering architectures.

Finally, we take the eff-5-freq architectures for the rd53_251, urf2_277, and f2_232 as an example to demonstrate the design process of the new architecture, in order to facilitate readers to better understand the workflow of our method, as shown in Fig. 7. Figure 7a shows the initial architecture set for filtering. It can be seen that the eff-5-freq architecture of urf2_277 has the largest number of edges, therefore, it serves as the target graph $\mathbf{G}^{\mathrm{T}}$ and the other two architectures serve as the pattern graph $\mathbf{G}_k^{\mathbf{P}}$, $k = 1, 2$. Both $|r|$ values are greater than 0.8, meeting our filtering criteria. Based on the improved GAPSO algorithm, we finally get the maximum common architecture $\mathbf{G}^{\mathrm{M}}$, as shown in Fig. 7b. Figure 7c shows the addition of remaining qubits and connections. According to our addition rules, the insertion positions are green vertices A, B, C and D. We add qubit and connection to different insertion positions and calculate the yield rate values. Finally, choose the architecture scheme with a higher yield rate value to output.

4.2.3 Comparison of frequency collisions

In this subsection, we compare the yield rates of four special-purpose architectures and two general-purpose architectures for each number of qubits, as shown in Fig. 8. The x-axis represents the architecture-number, and the y-axis is the simulation result of the yield rate.

It can be seen that the max 4-qubit bus scheme has the lowest yield rate among the six architectures, which means it is more prone to frequency collisions. This is due to that the qubits in this scheme have more connections. Having more connections on each qubit implies that each qubit is more likely to occur frequency collisions [33]. Therefore, the yield rate has decreased. Compared to the max 4-qubit bus scheme, the all 2-qubit bus scheme reduces the connections between qubits and thus has a higher yield rate. The yield rate of the all 2-qubit bus as shown in Fig. 5a is one order of

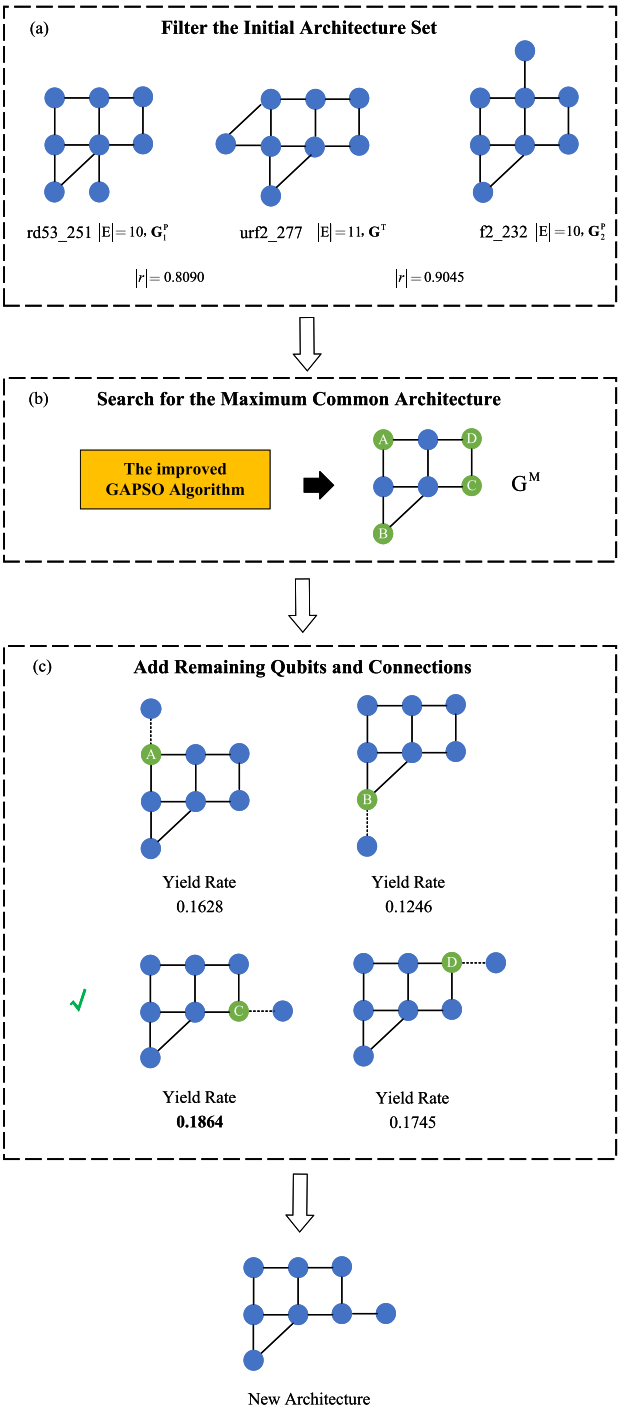

Fig. 7 Example of the method workflow

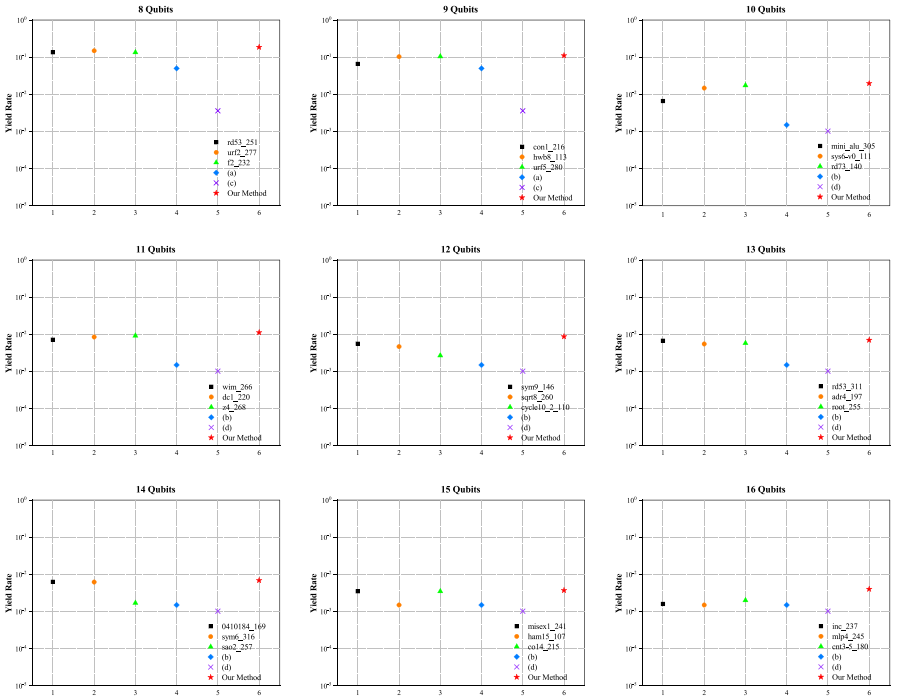

Fig. 8 Comparison of yield rate of different architectures

magnitude higher than that of the max 4-qubit bus as shown in Fig. 5c, and the all 2-qubit bus as shown in Fig. 5b is 1.47× higher than that of the max 4-qubit bus as shown in Fig. 5d.

The yield rates of the four special-purpose architectures are higher than or equal to the all 2-qubit bus scheme, which indicates the feasibility of designing special-purpose architectures to reduce frequency collisions. Moreover, among the four special-purpose architectures, the yield rates of our schemes are higher, with a minimum increase of 4.48% (in rd53_311) and a maximum increase of 3.06× (in sao2_257) compared to the eff-5-freq architectures. They also represent the percentage decrease in the probability of frequency collisions. This validates the effectiveness of the remaining qubits and connections addition strategy (Sect. 3.3).

4.2.4 Comparison of adaptability

We analyze the adaptability of our method by taking the architecture schemes generated by different special-purpose architecture design methods as input. Specifically, we conducted a further simulation comparison with the special-purpose superconducting quantum processor architecture design method of a recent work Ref. [34]. The method proposed in Ref. [34] is more effective for the program whose average degree of quantum program topology is less than or equal to 6. Compared to the quantum programs we selected, three 10-qubit quantum programs meet the conditions, and their average

Table 3 Average degree of three 10 Qubits quantum programs

	Program	Qubits number	Ave. degree
1	mini_alu_305	10	4.4
2	sys6-v0_111	10	4.8
3	rd73_140	10	4.8

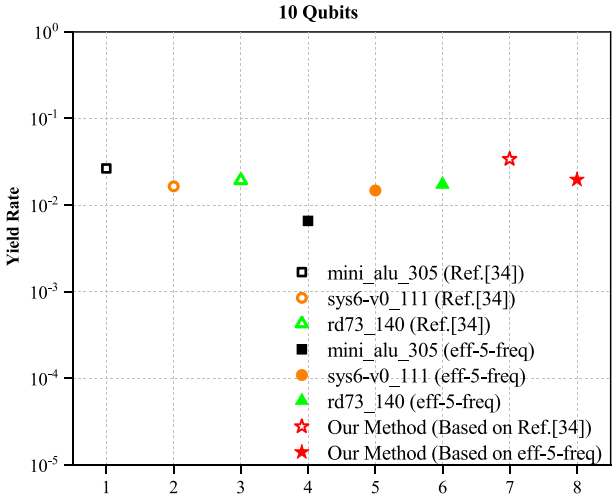

Fig. 9 Comparison of yield rate of different architectures in 10 Qubits

degrees are shown in Table 3. Therefore, we chose these three quantum programs for simulation comparison of architecture design methods.

Firstly, we design the special-purpose architectures for three quantum programs based on the architecture design method proposed in Ref. [34]. Secondly, based on three special-purpose architecture schemes, we use our method to generate the new architecture. Finally, simulation comparisons are conducted based on these architectures, including comparison of execution effects (Table 4) and frequency collisions (Fig. 9). In Table 4, we take the total post-mapping gate count of the special-purpose architecture of Ref. [34] as the baseline value. Same as Sect. 4.2.2, we compared the total post-mapping gate count changes in these architecture schemes (including architecture schemes related to the eff-5-freq). The x-axis and the y-axis in Fig. 7 have the same meanings as described in Sect. 4.2.3. Similarly, the comparison includes architectures related to the eff-5-freq method.

Through comprehensive simulation comparison, it can be seen that the architecture schemes (three 10-qubit quantum programs) designed by Ref. [34] are superior to the architecture schemes of the eff-5-freq. Compared to the architecture schemes in Ref. [34], the architecture schemes of the eff-5-freq method have an average increase of 4.4% in the total post-mapping gate count (run the corresponding quantum programs). Moreover, the yield rates in the architecture schemes in Ref. [34] are higher than those in the architecture schemes of the eff-5-freq, with a minimum increase of 10.92%

Table 4 Comparison of execution effects of different architectures in 10 Qubits

	Ref. [34]			eff-5-freq			Our Method (Based on Ref. [34])	Our Method (Based on eff-5-freq)
	mini_alu_305	sys6-v0_111	rd73_140	mini_alu_305	sys6-v0_111	rd73_140		
mini_alu_305	0%	11.76%	9.24%	12.18%	14.29%	23.95%	**7.56%**	12.18%
sys6-v0_111	5.78%	0%	12.59%	5.44%	1.02%	21.43%	**1.70%**	1.70%
rd73_140	10.58%	8.65%	0%	11.22%	9.62%	0%	**4.81%**	6.73%
$\|r\|$		0.9037	0.9223					
Avg	5.45%	6.80%	7.28%	9.61%	8.31%	15.13%	**4.69%**	6.87%

(in rd73_140). This demonstrates the feasibility of using graph theory to solve the performance and frequency collisions trade-off problem in Ref. [34].

By comparing the execution effects of different quantum programs, the architecture scheme of our method (based on Ref. [34]) has the smallest average growth in the total post-mapping gate count, which is 4.69%. Moreover, the higher $|r|$, the fewer gate operations added when the quantum program corresponding to $\mathbf{G}^{\mathrm{T}}$ runs on the $\mathbf{G}_k^{\mathrm{P}}$ architecture. Similarly, these demonstrate the rationality of designing the new architecture based on the maximum common structure and using Pearson correlation coefficient. In addition, in the compared architectures, the architecture scheme of our method (based on Ref. [34]) has a better yield rate, with a minimum improvement of 27.82%. Once again, it proves the effectiveness of the remaining qubits and connections addition strategy in Sect. 3.3. The comprehensive simulation comparison also proves that our method can be adapted to different special-purpose superconducting quantum processor architecture design methods.

Through the comprehensive comparison of different architecture simulation experiments in Sects. 4.2.2 and 4.2.3, one can conclude that the architecture schemes of our method can reduce more mapping overhead for multiple quantum programs, provide better execution support, and improve the reusability of special-purpose processors under the condition of ensuring the minimum impact of frequency collisions. As can be seen through Sect. 4.2.4, with the architecture schemes of different special-purpose architecture design methods as input, our method is able to generate architectures with better matching for multiple quantum programs. This verifies the reliability of our method.

5 Conclusion and outlook

In this paper, we propose a processor architecture design method based on graph theory to improve the reusability of special-purpose superconducting quantum processor. This method mainly constructs a new architecture based on the MCESs among multiple special-purpose processor architectures, including three steps: filtering the initial architecture set, searching for the maximum common structure, and adding remaining qubits and connections. Simulation results show that, under the comprehensive comparison of the average growth of the gate operations after qubit mapping and frequency collisions impact, the architecture schemes of our method are superior to the special-purpose architectures generated by the eff-5-freq algorithm and two general-purpose architectures based on the square lattice, which improves the reusability of special-purpose processors. Moreover, our method has a strong adaptability to generate architectures with better matching for multiple quantum programs when the architecture schemes of different special-purpose architecture design methods as input.

In addition, the strategies used in our method are not unique. For example, in the step of adding remaining qubits and connections, our adding rule is aimed at reducing the impact of frequency collisions. According to the actual situation, adding rules can also be considered from the perspective of improving the gate operations after qubit mapping. In future, we will consider more constraints and judgments, such as

crosstalk, fidelity and disconnected graph, to further optimize the special-purpose processor architecture.

Acknowledgements This work was supported by the Major Science and Technology Projects in Henan Province(CN), Grant No.: 221100-210400, 221100-210600.

Declarations

Conflict of interest The authors declare no Conflict of interest.

References

1. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. **26**(5), 1484–1509 (1997). https://doi.org/10.1137/S0097539795293172
2. Georgescu, I.M., Ashhab, S., Nori, F.: Quantum simulation. Rev. Mod. Phys. **86**, 153–185 (2014). https://doi.org/10.1103/RevModPhys.86.153
3. Kandala, A., Mezzacapo, A., Temme, K.: Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. Nature **549**(7671), 242–246 (2017). https://doi.org/10.1038/nature23879
4. Biamonte, J., Wittek, P., Pancotti, N., Rebentrost, P., Wiebe, N., Lloyd, S.: Quantum machine learning. Nature **549**(7671), 195–202 (2017)
5. Krantz, P., Kjaergaard, M., Yan, F., Orlando, T., Gustavsson, S., Oliver, W.: A quantum engineer's guide to superconducting qubits. Appl. Phys. Rev. **6**(2), 021318 (2019)
6. Murali, P., Debroy, D.M., Brown, K.R., Martonosi, M.: Toward systematic architectural design of near-term trapped ion quantum computers. Commun. ACM **65**(3), 101–109 (2022). https://doi.org/10.1145/3511064
7. He, Y., Gorman, S., Keith, D., Kranz, L., Keizer, J., Simmons, M.: A two-qubit gate between phosphorus donor electrons in silicon. Nature **571**(7765), 371–375 (2019). https://doi.org/10.1038/s41586-019-1381-2
8. Wang, H., Qin, J., Ding, X., Chen, M.-C., Chen, S., You, X., He, Y.-M., Jiang, X., You, L., Wang, Z., Schneider, C., Renema, J.J., Höfling, S., Lu, C.-Y., Pan, J.-W.: Boson sampling with 20 input photons and a 60-mode interferometer in a 10^{14}-dimensional Hilbert space. Phys. Rev. Lett. **123**, 250503 (2019). https://doi.org/10.1103/PhysRevLett.123.250503
9. Huang, H.-L., Wu, D., Fan, D., Zhu, X.: Superconducting quantum computing: a review. Sci. China Inf. Sci. **63**, 1–32 (2020). https://doi.org/10.1007/s11432-020-2881-9
10. Alt, R.: On the potentials of quantum computing-an interview with Heike Riel from IBM research. Electron. Mark. **32**(4), 2537–2543 (2022). https://doi.org/10.1007/s12525-022-00616-1
11. Ai, G.Q.: Suppressing quantum errors by scaling a surface code logical qubit. Nature **614**(7949), 676–681 (2023)
12. Arute, F., Arya, K., Babbush, R., Bacon, D., Bardin, J.C., Barends, R., Biswas, R., Boixo, S., Brandao, F.G., Buell, D.A., et al.: Quantum supremacy using a programmable superconducting processor. Nature **574**(7779), 505–510 (2019). https://doi.org/10.1038/s41586-019-1666-5
13. Gong, M., Wang, S., Zha, C., Chen, M.-C., Huang, H.-L., Wu, Y., Zhu, Q., Zhao, Y., Li, S., Guo, S., et al.: Quantum walks on a programmable two-dimensional 62-qubit superconducting processor. Science **372**(6545), 948–952 (2021)
14. Chow, J., Dial, O., Gambetta, J.: IBM quantum breaks the 100-qubit processor barrier. IBM Research Blog (2021)
15. Hugh, C.: IBM unveils 400 qubit-plus quantum processor and next-generation IBM quantum system two. IBM Research Blog (2022)
16. Place, A.P.M., Rodgers, L.V.H., Mundada, B.M., Smitham, P., Fitzpatrick, M., Leng, Z., Premkumar, A., Bryon, J., Vrajitoarea, A., Sussman, S., et al.: New material platform for superconducting transmon qubits with coherence times exceeding 0.3 milliseconds. Nat. Commun. **12**(1), 1–6 (2021)
17. Wang, C., Li, X., Xu, H., Li, Z., Wang, J., Yang, Z., Mi, Z., Liang, X., Su, T., Yang, C., et al.: Towards practical quantum computers: Transmon qubit with a lifetime approaching 0.5 milliseconds. npj Quantum Inf. **8**(1), 3 (2022). https://doi.org/10.1038/s41534-021-00510-2

18. Sheldon, S., Magesan, E., Chow, J.M., Gambetta, J.M.: Procedure for systematically tuning up cross-talk in the cross-resonance gate. Phys. Rev. A **93**(6), 060302 (2016). https://doi.org/10.1103/PhysRevA.93.060302
19. Kirchhoff, S., Keßler, T., Liebermann, P.J., Assémat, E., Machnes, S., Motzoi, F., Wilhelm, F.K.: Optimized cross-resonance gate for coupled transmon systems. Phys. Rev. A **97**(4), 042348 (2018). https://doi.org/10.1103/PhysRevA.97.042348
20. Stehlik, J., Zajac, D.M., Underwood, D.L., Phung, T., Blair, J., Carnevale, S., Klaus, D., Keefe, G.A., Carniol, A., Kumph, M., Steffen, M., Dial, O.E.: Tunable coupling architecture for fixed-frequency transmon superconducting qubits. Phys. Rev. Lett. **127**(8), 080505 (2021). https://doi.org/10.1103/PhysRevLett.127.080505
21. Li, S., Fan, D., Gong, M., Ye, Y., Chen, X., Wu, Y., Guan, H., Deng, H., Rong, H., Huang, H.-L., et al.: Realization of fast all-microwave controlled-z gates with a tunable coupler. Chin. Phys. Lett. **39**(3), 030302 (2022). https://doi.org/10.1088/0256-307X/39/3/030302
22. Li, G., Wu, A., Shi, Y., Javadi-Abhari, A., Ding, Y., Xie, Y.: On the co-design of quantum software and hardware. In: Proceedings of the Eight Annual ACM International Conference on Nanoscale Computing and Communication, pp. 1–7. Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3477206.3477464
23. Linke, N.M., Maslov, D., Roetteler, M., Debnath, S., Figgatt, C., Landsman, K.A., Wright, K., Monroe, C.: Experimental comparison of two quantum computing architectures. Proc. Natl. Acad. Sci. **114**(13), 3305–3310 (2017). https://doi.org/10.1073/pnas.1618020114
24. 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: Proceedings of the 46th International Symposium on Computer Architecture, pp. 527–540. Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3307650.3322273
25. Hu, W., Yang, Y., Xia, W., Pi, J., Huang, E., Zhang, X.-D., Xu, H.: Performance of superconducting quantum computing chips under different architecture designs. Quantum Inf. Process. **21**(7), 1–14 (2022)
26. Hillmich, S., Zulehner, A., Wille, R.: Exploiting quantum teleportation in quantum circuit mapping. In: Proceedings of the 26th Asia and South Pacific Design Automation Conference, pp. 792–797. Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3394885.3431604
27. Brink, M., Chow, J.M., Hertzberg, J., Magesan, E., Rosenblatt, S.: Device challenges for near term superconducting quantum processors: frequency collisions. In: 2018 IEEE International Electron Devices Meeting (IEDM), pp. 1–3. IEEE, San Francisco, CA, USA (2018)
28. Chamberland, C., Zhu, G., Yoder, T.J., Hertzberg, J.B., Cross, A.W.: Topological and subsystem codes on low-degree graphs with flag qubits. Phys. Rev. X **10**, 011022 (2020). https://doi.org/10.1103/PhysRevX.10.011022
29. Dallaire-Demers, P.-L., Wilhelm, F.K.: Quantum gates and architecture for the quantum simulation of the fermi-Hubbard model. Phys. Rev. A **94**(6), 062304 (2016). https://doi.org/10.1103/PhysRevA.94.062304
30. Li, G., Shi, Y., Javadi-Abhari, A.: Software-hardware co-optimization for computational chemistry on superconducting quantum processors. In: 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA), pp. 832–845. ACM/IEEE, Virtual Event, Spain (2021)
31. Deb, A., Dueck, G.W., Wille, R.: Exploring the potential benefits of alternative quantum computing architectures. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **40**(9), 1825–1835 (2021). https://doi.org/10.1109/TCAD.2020.3032072
32. Lin, W.-H., Tan, B., Niu, M.Y., Kimko, J., Cong, J.: Domain-specific quantum architecture optimization. IEEE J. Emerg. Sel. Top. Circuits Syst. **12**(3), 624–637 (2022). https://doi.org/10.1109/JETCAS.2022.3202870
33. Li, G., Ding, Y., Xie, Y.: Towards efficient superconducting quantum processor architecture design. In: Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 1031–1045 (2020)
34. Yang, T., Wang, W., Wang, L., Zhao, B., Liang, C., Shan, Z.: A superconducting quantum processor architecture design method for improving performance and reducing frequency collisions. Results Phys. **53**, 106944 (2023). https://doi.org/10.1016/j.rinp.2023.106944

35. Huang, R., Geng, X., Wu, X., Dai, G., Yang, L., Liu, J., Chen, W.: Cryogenic multiplexing control chip for a superconducting quantum processor. Phys. Rev. Appl. **18**(6), 064046 (2022). https://doi.org/10.1103/PhysRevApplied.18.064046
36. Gold, A., Paquette, J., Stockklauser, A., Reagor, M.J., Alam, M.S., Bestwick, A., Didier, N., Nersisyan, A., Oruc, F., Razavi, A., et al.: Entanglement across separate silicon dies in a modular superconducting qubit device. npj Quantum Inf. **7**(1), 142 (2021). https://doi.org/10.1038/s41534-021-00484-1
37. Zhong, Y., Chang, H.-S., Bienfait, A., Dumur, É., Chou, M.-H., Conner, C.R., Grebel, J., Povey, R.G., Yan, H., Schuster, D.I., et al.: Deterministic multi-qubit entanglement in a quantum network. Nature **590**(7847), 571–575 (2021). https://doi.org/10.1038/s41586-021-03288-7
38. Niu, J., Zhang, L., Liu, Y., Qiu, J., Huang, W., Huang, J., Jia, H., Liu, J., Tao, Z., Wei, W., et al.: Low-loss interconnects for modular superconducting quantum processors. Nat. Electron. (2023). https://doi.org/10.1038/s41928-023-00925-z
39. Bahiense, L., Manić, G., Piva, B., De Souza, C.C.: The maximum common edge subgraph problem: a polyhedral investigation. Discrete Appl. Math. **160**(18), 2523–2541 (2012). https://doi.org/10.1016/j.dam.2012.01.026
40. Cohen, I., Huang, Y., Chen, J., Benesty, J., Benesty, J., Chen, J., Huang, Y., Cohen, I.: Pearson correlation coefficient. In: Noise Reduction in Speech Processing, pp. 1–4 (2009)
41. Liu, X., Qu, H., Meng, L., Chen, Q., Wang, C., Wang, Q.: Sensorless control of the BPMSM for artificial heart based on the improved SMO, the improved HFI and the GAPSO algorithm. Measurement **207**, 112305 (2023). https://doi.org/10.1016/j.measurement.2022.112305
42. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning, 1st edn. Addison-Wesley Longman Publishing Co. Inc, New York (1989)
43. Poli, R., Kennedy, J., Blackwell, T.: Particle swarm optimization: an overview. Swarm Intell. **1**, 33–57 (2007)
44. 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, pp. 1–6. Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3316781.3317859
45. 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. Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3168822
46. Li, G., Ding, Y., Xie, Y.: Tackling the qubit mapping problem for NISQ-ERA quantum devices. In: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 1001–1014. Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3297858.3304023
47. Hertzberg, J.B., Zhang, E.J., Rosenblatt, S., Magesan, E., Smolin, J.A., Yau, J.-B., Adiga, V.P., Sandberg, M., Brink, M., Chow, J.M., et al.: Laser-annealing Josephson junctions for yielding scaled-up superconducting quantum processors. npj Quantum Inf. **7**(1), 1–8 (2021)
48. Zhang, E.J., Srinivasan, S., Sundaresan, N., Bogorin, D.F., Martin, Y., Hertzberg, J.B., Timmerwilke, J., Pritchett, E.J., Yau, J.-B., Wang, C., et al.: High-performance superconducting quantum processors via laser annealing of transmon qubits. Sci. Adv. **8**(19), 6690 (2022). https://doi.org/10.1126/sciadv.abi6690