Qubit Allocation for Distributed Quantum Computing

1st Yingling Mao
Dept. of Electrical and Computer Eng.
Stony Brook University
New York, USA
yingling.mao@stonybrook.edu

2nd Yu Liu
Dept. of Electrical and Computer Eng.
Stony Brook University
New York, USA
yu.liu.3@stonybrook.edu

3rd Yuanyuan Yang
Dept. of Electrical and Computer Eng.
Stony Brook University
New York, USA
yuanyuan.yang@stonybrook.edu

IEEE INFOCOM 2023 - IEEE Conference on Computer Communications | 979-8-3503-3414-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/INFOCOM53939.2023.10228915

***Abstract*—With the advancements in quantum communication, optically connected quantum processors can form a distributed quantum computing system. Distributed quantum computing provides a scalable path to execute more complicated computational tasks that a single quantum processor cannot handle. Yet, distributed quantum computing needs a new compiler to map logical qubits of a quantum circuit to different quantum processors in the system. This paper formulates and studies the qubit allocation problem for distributed quantum computing (QA-DQC). We prove the NP-hardness of the formulated problem. Moreover, we show there is no polynomial-time n^a-approximation algorithm for any $a < 1$ unless $P = NP$, where n is the number of processors in the quantum network. We first propose a heuristic local search algorithm for QA-DQC. Furthermore, we design a multistage hybrid simulated annealing algorithm (MHSA) by combining the local search algorithm and a simulated annealing meta-heuristic algorithm. Lastly, we perform extensive simulations to evaluate the proposed MHSA under various real quantum circuits and different network topologies. Results show that MHSA outperforms popular baselines.**

I. INTRODUCTION

Quantum computing can solve problems that are probably impossible or highly inefficient on classical servers. For example, Shor's algorithm can factorize integers around exponentially faster than algorithms on conventional servers [1], which gives Shor's algorithm the potential to break public-key cryptography schemes such as RSA. Many governments have invested enormous funds in quantum computing research in recent years. For example, the US Government signed a law that allows a $1.2 billion investment in quantum information [2], and the European Commission approved a $1-billion program to support quantum research [3]. In addition, large companies like IBM and Google are also competing in building commercial quantum computers. There have been lots of achievements in quantum computing in recent years. Google AI Quantum has achieved quantum supremacy, where performing calculations on their Sycamore quantum computer is 3,000,000 times faster than on a classical supercomputer [4]. IBM has presented a 127-Qubit (the quantum analog of the classical bit) Quantum Processor named IBM Eagle in November 2021, which any classical computer can access [5].

Despite the benefits and achievements of quantum computing, building a single large processor in terms of the number of qubits is challenging. One difficulty in building quantum processors is quantum decoherence due to qubits interacting with their environments, leading to system errors. The more qubits there are, the more system errors. As the number of qubits in a processor increases, the classical resource required by the processor increases exponentially [6]. Practical applications of quantum computers require thousands, or even millions, of physical qubits, so it will be challenging for individual quantum processors to reach such qubit numbers. For example, when it comes to the unique shortest vector problem in cryptography [12], the circuit grows large: lattice dimension 3 already requires 842 qubits, which is far larger than the current biggest quantum processor capacity.

Thanks to the advancement of quantum communication [7], it is promising to build a distributed large-scale quantum processor by connecting different processors of relatively small scales [8]. We can perform gates on two qubits in different processors by exploiting quantum entanglements between the two processors (details can be found in Section III). A quantum state can be in a superposition of many states, which is exponential to the number of qubits. Therefore, as the number of connected processors increases, the computational power of the distributed quantum computer increases exponentially.

One critical problem when realizing distributed quantum computing is qubit allocation. Specifically, quantum circuits manipulate qubits, which are called logical qubits, since they exist as abstractions within a quantum circuit. Qubit allocation refers to the problem of mapping the logical qubits of a quantum circuit into physical qubits, which are the actual hardware units that store quantum bits. The qubit allocation problem in a single quantum processor has been solved by works [9], [10]. Besides, a mature qubit allocator called ibmmapper has been proposed as part of IBM's compiler and runtime infrastructure. But these techniques are all limited to the qubit allocation in a single quantum processor. The qubit allocation in distributed quantum computing is different due to quantum communication between different processors. So far, few works have been devoted to the qubit allocation problem in distributed quantum computing. At the same time, they are all limited to some special quantum circuits, specific quantum processors, or particular quantum network topologies.

In this paper, we focus on a general Qubit Allocation problem for Distributed Quantum Computing (QA-DQC), which

can adapt all possible quantum circuits, quantum processors, and quantum network topologies. Since local qubit allocation has been successfully solved, in our model of QA-DQC, we mainly focus on how to map the logical qubits of a quantum circuit to different processors. The goal of QA-DQC is to minimize the cost of remote operations between different processors, which is the overhead of quantum communication in the network. After determining the map of logical qubits and quantum processors, the existing local qubit allocation techniques can solve the rest local allocation problems, i.e., mapping logical qubits allocated to a quantum processor to the physical qubits of the processor.

Before solving QA-DQC, we first prove its NP-hardness. In addition, we show that there is no polynomial-time n^a-approximation algorithm for any $a < 1$ unless $P = NP$. Here n is the number of processors in the distributed quantum computing system. Since quantum gates accumulate noise, the compilation of quantum circuits in distributed quantum computing demands high compiler optimization precision. It implies that even the lowest approximation ratio, i.e., $O(n)$, is still far from reaching the precision requirement of real-world compilers in QA-DQC. Thus, in the design of our heuristic algorithm, we pay more attention to pursuing better average performance rather than the worst-case bound. To pursue great optimization performance, we take meta-heuristic algorithms into consideration. We choose a meta-heuristic algorithm called simulated annealing (SA). To further improve optimization performance, we design a heuristic local search algorithm for QA-DQC and combine it with SA to generate a multistage hybrid simulated annealing algorithm (MHSA), which has better performance than isolated SA or heuristic local search algorithm.

Our main contributions are listed as follows.

- We formulate the QA-DQC problem, which can adapt all possible quantum circuits, quantum processors, and quantum network topologies.
- We prove the NP-hardness of QA-DQC and further prove there is no polynomial-time n^a-approximation algorithm for any $a < 1$ unless $P = NP$, where n is the number of processors in the quantum network.
- We design an MHSA algorithm for QA-DQC, which combines a meta-heuristic algorithm called simulated annealing and a heuristic local search algorithm designed for QA-DQC.
- We perform extensive simulations on various real quantum circuits and different network topologies, demonstrating that MHSA outperforms popular baselines.

The remainder of this paper is organized as follows. Section II reviews the related works. In Section III, we give some background knowledge on quantum computing. Section IV states the architecture of distributed quantum computing.Based on the architecture, we formulate the QA-DQC problem in Section V. Afterward, Section VI a MHSA algorithm for QA-DQC. Then Section VII is the performance evaluation of MHSA. Finally, we conclude the paper in Section VIII.

II. RELATED WORK

Quantum computing [11] harnesses quantum mechanics to gain the ability to solve problems that are too complex for classical computers. The complexity of these targeted problems, like the unique shortest vector problem in cryptography [12], typically demands many physical qubits but building a monolithic quantum system with lots of qubits has technological limitations. These limitations are one of the reasons for moving toward distributed quantum computing [13].

In distributed quantum computing, different quantum processors are connected to generate more computational power than an isolated processor. But the communication between different processors produces overhead in the compiler of quantum circuits. Thus, the qubit allocation problem becomes a hot spot in distributed quantum computing. Because the concept of distributed quantum computing is so recent, so is the interest in the qubit allocation problem for distributed quantum computing. To the best of our knowledge, there have been these previous attempts [14]–[16], [18]–[20] to solve this problem.

Among them, three works [14]–[16] have a similar main idea, which is to formulate the QA-DQC problem as a graph partition problem. Zomorodi-Moghadam et al. [14] propose a general approach, based on the Kernighan-Lin algorithm for graph partitioning, to optimize the number of teleportations for a distributed quantum computing architecture consisting of two spatially separated and long-distance quantum subsystems. Andres-Martinez et al. [15] develop an automated method to distribute quantum circuits, which turns the quantum circuit into a hypergraph, then finds a partitioning utilizing the KaHyPar solver [17] to minimize the number of cuts. Davarzani et al. [16] put forward a dynamic programming algorithm to reduce the number of communications in a distributed quantum circuit. But such a basic idea and these proposed solutions have some drawbacks. In particular, they ignore the influence of quantum network topology on the overhead, which is unpractical. We consider quantum network graph and formulate QA-DQC as a generalization problem of Quadratic Assignment Problem.

Besides, the other existing related works try to design approximation algorithms for QA-DQC and achieve some performance guarantee. Yimsiriwattana et al. [18] present a distributed implementation of Shor's quantum factoring algorithm on integer N on a distributed quantum network model and prove this distributed version of Shor's algorithm requires an additional overhead of $O((logN)^2)$ communication complexity. Beals et al. [19] provide approximation algorithms for efficiently moving and addressing quantum memory in parallel, enabling the standard circuit model to be simulated with a low overhead by a more realistic model of a distributed quantum computer (DQC). And they prove as for the DQC model with graph G, N processors, and $O(\log N)$ qubits per processor, the overhead can be bounded by $O(N)$. Ferrari et al. [20] derive an upper bound $O(n)$ of the overhead induced

by quantum compilation for distributed quantum computing in the network of linear topology, where n denotes the number of logical qubits. However, they have two main shortcomings: One is that the proven approximation ratios are too large to satisfy the demand for extremely accurate compiler optimization in distributed quantum computing; the other is that they are all limited to some special quantum circuits, special quantum network topologies, or special quantum processors. Different from them, we design a meta-heuristic-based algorithm, which can solve QA-DQC with any quantum circuits, quantum processors, and network topologies. Besides, the meta-heuristic-based algorithm has a performance advantage, adapting to high compiler optimization precision demands in QA-DQC.

III. Background

A. Qubit

In quantum computing, a qubit, short for a quantum bit, is a basic unit of quantum information, like the binary digit in classical computers. A qubit is a two-state quantum-mechanical system. In a classical system, the binary digit must be either 0 or 1. However, quantum mechanics allows the qubit to be in a coherent superposition of both states simultaneously, which is a fundamental property of quantum computing. We can use two orthogonal basis vectors of a 2D complex vector space, $|0\rangle = \binom{1}{0}$ and $|1\rangle = \binom{0}{1}$, to represent the two basic states of a qubit, then the state of a single qubit can be described by a linear combination of $|0\rangle$ and $|1\rangle$:

$$\alpha\,|0\rangle + \beta\,|1\rangle = \alpha\binom{1}{0} + \beta\binom{0}{1} = \binom{\alpha}{\beta},$$

where α and β are complex numbers and $\|\alpha\|^2 + \|\beta\|^2 = 1$. When we measure this qubit in the standard basis of $|0\rangle$ and $|1\rangle$, according to the Born rule, the probability of outcome $|0\rangle$ is $\|\alpha\|^2$, and the probability of outcome $|1\rangle$ is $\|\beta\|^2$.

B. Quantum Gate and Circuit

Quantum logic gates, simply called quantum gates, are basic quantum operators on qubits to change their states. For example, in the standard basis, the *Hadamard-Wash* gate H can be represented by unitary matrix $\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$ and maps the basis state $|0\rangle$ to $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$, $|1\rangle$ to $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$. Similarly, any single-qubit gates, i.e., the gates applied on a single qubit, can be represented as a 2×2 matrix.

The CNOT, short for Controlled Not, gate applies on two qubits. $CNOT(q_1, q_2)$ indicates that qubit q_1 control qubit q_2 by negating q_2 if and only if q_1 is $|1\rangle$. Specifically,

$$CNOT(q_1, q_2) = (q_1, q_1 \bigoplus q_2).$$

Quantum gates are also the building blocks of quantum circuits like classical logic gates are for conventional digital circuits. Fig. 1 shows a quantum circuit, which implements 2-qubit Glover's search algorithm for searching $|00\rangle$. This circuit has two logical qubits q_0, q_1, which are represented as horizontal lines. It uses three different types of quantum gates, i.e., H, X, and CNOT, and the final two measurements to operate these two qubits.

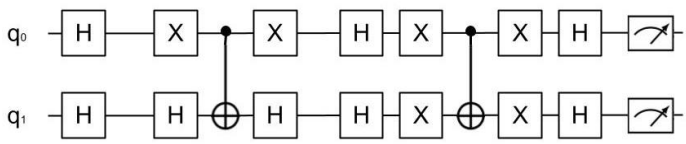


Fig. 1: Quantum circuit of 2-qubit Glover's search algorithm

Work [21] shows all single-qubit gates and the CNOT gate form a universal set of gates that can implement arbitrary circuits.

C. Quantum Processor

In practice, quantum processors based on superconducting qubits (or NV center qubits) technology are modes of solid-state circuits that only allow local interactions between the physical qubits are connected together [22], [23]. Technical reasons restrict the number of physical qubits, the number of coupling connections, and their organization topology. [24] So far, the biggest quantum processor is IBM's Eagle processor, which packs 127 qubits in its quantum-computing chip.

In a quantum processor, the physical qubits are divided into two kinds: the *computation qubits*, devoted to running the quantum circuit, and the *communication qubits*, used for communication between processors.

D. Quantum Network / Cluster

A quantum network facilitates information transmission in the form of qubits between physically separated quantum processors. It works in a similar way to the classical network. The main difference is the communication method based on quantum entanglement.

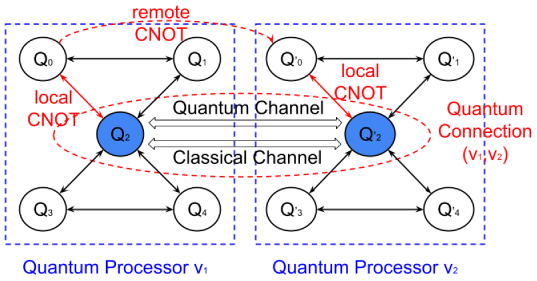
Quantum entanglement is the physical phenomenon that occurs when a group of particles shares spatial proximity in a way that the quantum state of each particle of the group cannot be described independently of the state of the others, even when the particles are separated by a large distance. In quantum network communication, we only employ a special case of maximally entangled two-qubit states called EPR (short for Einstein-Podolsky-Rosen) pairs. On the standard basis, the states of EPR pairs can be written as $|\Phi^+\rangle = \frac{|00\rangle+|11\rangle}{\sqrt{2}}$.

In a quantum network /cluster, which is represented by a graph $H = (V, E)$, two processors $v_1, v_2 \in V$ can communicate directly only when there is a classical channel, a quantum channel, and at least one pair of communication qubits between them. If so, we consider these two quantum processors are connected directly, i.e. $(v_1, v_2) \in E$. Note that the quantum channel here is used for EPR pair generation.

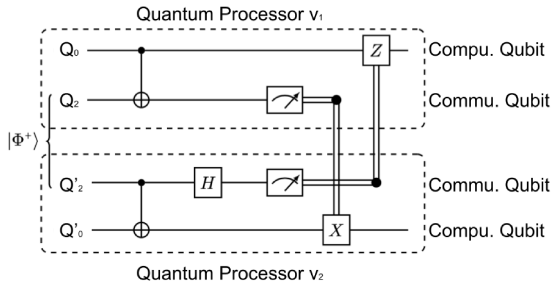
IV. Distributed Quantum Computing

A. The Architecture of Distributed Quantum Computing

Distributed quantum computing can leverage the power of interconnected quantum processors to create a virtual quantum machine with processing capabilities that surpass its physical constituents alone. It is analogous to connecting several



(a) A virtual quantum processor built by a 2-node cluster. Here two quantum processors v_1, v_2 are both IBM Yorktown quantum processors with 5 physical qubits. Letting Q_2, Q_2' work as the communication qubits, we can build the quantum connection between v_1 and v_2. Based on this quantum connection, these two 5-qubit processors constitute an 8-qubit virtual quantum processor, where qubits interact via local and remote CNOT.



(b) The circuit for the remote CNOT operation between the two computation qubits Q_0, Q_0' from two different but directedly connected quantum processors v_1, v_2. Here Q_2, Q_2' are the pair of communication qubits for the connection (v_1, v_2). Based on this connection, an EPR pair $|\Phi^+\rangle$ can be generated on Q_2, Q_2'. The double line denoted the transmission of one bit of classical information, i.e., the measurement outcome, between two processors v_1, v_2.

Fig. 2: Toy-model for distributed quantum computing of 2-node cluster

classical computers to form a computer cluster in classical computing. Distributed quantum computing works by linking multiple Less powerful quantum processors through quantum connections to create one more powerful virtual quantum processor, or called a quantum computing cluster. For example, Fig. 2a shows a virtual quantum processor built by linking two 5-qubit IBM Yorktown quantum processors v_1, v_2 by a quantum connection based on the pair of communication qubits Q_2, Q_2'. It can also be called a 2-node quantum computing cluster. We can see the constituted cluster has $5 + 5 - 2 = 8$ computation qubits by subtracting the two communication qubits, having more computation power than each component processor. Like classical computing, this system is scalable by adding more and more quantum processors to the cluster.

Like classical distributed computing, an essential requirement for distributed quantum computing is the possibility of remote operations, the operations between logical qubits stored at different processors. Recall arbitrary quantum circuits can be built based on the CNOT gate and all single-qubit gates. Since a single-qubit gate can be implemented locally on a single computation qubit of any processor, we focus on the possibility of the remote CNOT gates, like $CNOT(Q_0, Q_0')$ in Fig. 2a. Below we will show how remote CNOT works based on a pair of maximally entangled communication qubits between two directly connected quantum processors (Seen in Section IV-B) and further how remote CNOT works via a routing path between two remote quantum processors (Seen in Section IV-C).

B. Remote CNOT via A Quantum Connection

A quantum connection means a classical channel, a quantum channel, and at least one pair of communication qubits. For example, in Fig. 2a, there is a quantum connection (v_1, v_2) between quantum processor v_1, v_2 and the pair of communication qubits are Q_2, Q_2'. Then based on this quantum connection, an EPR pair $|\Phi^+\rangle$ can be generated on the pair of communication qubits Q_2, Q_2'. Now, Q_2, Q_2' are maximally entangled and their current state is exactly $|\Phi^+\rangle = \frac{|00\rangle+|11\rangle}{\sqrt{2}}$.

Work [25], [26] shows the claim that *if there is a pair of maximally entangled communication qubits between two quantum processors, remote CNOT can be successfully implemented on any pair of computation qubits from these two quantum processors.* Fig. 2b shows the circuit for the remote CNOT operation between the two computation qubits Q_0, Q_0' in Fig. 2a. After an EPR pair is available on the pair of communication qubits Q_2, Q_2', we can implement remote CNOT on Q_0, Q_0' through a local CNOT on (Q_0, Q_2) and (Q_2', Q_0') at each processor, followed by some shown single-qubit gates and measurements. Note that the double line shown in the figure does not mean a two-qubit operation like CNOT. It is the transmission of one bit of classical information, i.e., the measurement outcome, between two processors v_1, v_2. And X, Z are both single-qubit gates controlled by the received measurement outcome, i.e., X, Z is implemented only when the received classical digit is 1; if not, keep the state as it is.

Below, we will give a brief proof to show that the circuit in Fig. 2b realizes the remote CNOT on Q_0, Q_0'. Assume $Q_0 = \alpha|0\rangle + \beta|1\rangle$, $Q_0' = \alpha'|0\rangle + \beta'|1\rangle$, by CNOT operation, the state of $Q_0 Q_0'$ should be

$$|\phi\rangle = \alpha|0\rangle(\alpha'|0\rangle + \beta'|1\rangle) + \beta|1\rangle(\alpha'|1\rangle + \beta'|0\rangle).$$

According to the circuit in Fig. 2b, the initial state is $Q_0|\Phi^+\rangle Q_0'$. After two local CNOT on (Q_0, Q_2) and (Q_2', Q_0') at each processor, the state of $Q_0 Q_2 Q_2' Q_0'$ should be

$$\begin{aligned}|\phi_1\rangle =& \frac{\alpha}{\sqrt{2}}|0\rangle\left[|00\rangle(\alpha'|0\rangle + \beta'|1\rangle) + |11\rangle(\alpha'|1\rangle + \beta'|0\rangle)\right] \\ &+ \frac{\beta}{\sqrt{2}}|1\rangle\left[|10\rangle(\alpha'|0\rangle + \beta'|1\rangle) + |01\rangle(\alpha'|1\rangle + \beta'|0\rangle)\right].\end{aligned}$$

If the measure of Q_2 is 0, X is not implemented on Q_0', now the state of $Q_0 Q_2' Q_0'$ is

$$|\phi_2\rangle = \frac{\alpha}{\sqrt{2}}|00\rangle(\alpha'|0\rangle + \beta'|1\rangle) + \frac{\beta}{\sqrt{2}}|11\rangle(\alpha'|1\rangle + \beta'|0\rangle).$$

If the measure of Q_2 is 1, X is implemented on Q_0', so

$$|\phi_2\rangle = \frac{\alpha}{\sqrt{2}}|01\rangle(\alpha'|0\rangle + \beta'|1\rangle) + \frac{\beta}{\sqrt{2}}|10\rangle(\alpha'|1\rangle + \beta'|0\rangle).$$

Now we can see in two cases of $|\phi_2\rangle$, the states of Q_0Q_0' are both exactly what we want ($|\phi\rangle$). Now our task is to collapse Q_2' without changing the state of Q_0Q_0'. It can be successfully done by the Hadamard gate on Q_2', the followed measurement, and Z gate controlled by the measurement outcome on Q_0.

In all, if two quantum processors are directly connected, a pair of maximally entangled communication qubits can be generated through quantum connection and helps implement remote CNOT between two processors.

C. Remote CNOT via A Routing Path

In order to implement the remote CNOT between the two processor nodes that are not directly connected, the key is to build a pair of maximally entangled communication qubits between them.

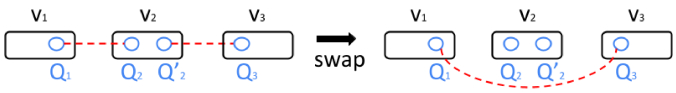


Fig. 3: An example of entanglement swapping.

The technology of entanglement swapping can help us make it. Fig. 3 presents an example of entanglement swapping, where the red dotted lines represent a pair of maximally entangled qubits. In the figure, we can see processors v_1 and v_2 share an EPR pair Q_1, Q_2 while processors v_2 and v_3 share an EPR pair Q_2', Q_3. Then we can make an entanglement swapping by running a Bell-state measurement circuit on the two communication qubits Q_2, Q_2' at processor Q_2 so that communication qubits Q_1, Q_3 become an EPR pair, a pair of maximally entangled qubits [27]. Suppose there are more than one mid-nodes, like processor node v_2. In that case, we can make simultaneous entanglement swapping by simultaneously running a Bell-state measurement circuit at each mid-processor node so that an EPR pair appears between the starting and ending nodes [28].

In all, we can implement the remote CNOT between two remote quantum processors by the following steps:

- Step 1: Find the shortest routing path T between the two targeted processor nodes on the graph of the cluster;
- Step 2: Generate an EPR pair for each quantum connection that belongs to the path T;
- Step 3: Simultaneously run a Bell-state measurement circuit at each mid-processor node on the path T;
- Step 4: Implement the circuit for the remote CNOT between the two targeted processors, like Fig. 2b.

V. Qubit Allocation

A. Problem Formulation

This section will formulate the qubit allocation problem for distributed quantum computing. Specifically, decide how to allocate the logical qubits of the targeted quantum circuit to the different processors in a quantum computing cluster.

As we mentioned in Section IV, we need additional operations for communication, such as the EPR pair generation, the Bell-state measurement circuit, and the remote CNOT circuit, to realize the remote operation (CNOT) between different processors in a quantum computing cluster. However, in quantum computing, decoherence and noise effects severely constrain the execution time and the operation procedures. Unlike classical digital gates that are inherently self-stabilizing, quantum gates accumulate noise. Although quantum error-correcting codes (QEC) hold the promise to address decoherence issues [29], current hardware does not provide nearly enough resources to implement realistic QEC [30]. The longer a quantum program runs and the more operations it performs, the more it is susceptible to noise. Therefore, minimizing the communication overhead, i.e., minimizing the total cost of the remote CNOT gates, is crucial. It affects not only the real running time of quantum programming but also the accuracy of the quantum circuit. For these reasons, the compilation of quantum circuits in distributed quantum computing demands extremely accurate compiler optimization with the objective of minimizing the total cost of the remote CNOT gates (communication overhead).

The input to the optimization problem consists of the following:

1) The network graph of the quantum computing cluster $H = (V, E)$, where each node $v \in V$ corresponds to quantum processors and each edge $e = (v_1, v_2) \in E$, where $v_1, v_2 \in V$, corresponds to quantum connections.
2) A set of logical qubits Q and the set of CNOT gates G, where each CNOT gate $g = CNOT(q_1, q_2) \in G$ operates on a pair of qubits $q_1, q_2 \in Q$. Note that $CNOT(q_1, q_2) \neq CNOT(q_2, q_1)$.
3) The EPR pair generation cost C_{epr}, the cost of a Bell-state measurement circuit C_{bsm} and the cost of a remote CNOT circuit C_{rcn}.

For all $q \in Q$ and $v \in V$, we define a binary variable $x_{q,v} \in \{0, 1\}$, which takes the value 1 if and only if the logical qubit $q \in Q$ resides at the node $v \in V$. To make sure that each logical qubit is located at exactly one node, we have the following constraint:

$$\sum_{v \in V} x_{q,v} = 1, \quad \forall q \in Q. \quad \text{(qubit)} \tag{1}$$

In addition, since the quantum processor $v \in V$ has n_v computation qubits, we have the following constraint to make sure that node v can host at most n_v logical qubits:

$$\sum_{q \in Q} x_{q,v} \leq n_v, \quad \forall v \in V. \quad \text{(node)} \tag{2}$$

For all $q_1, q_2 \in Q$, we denote the number of CNOT gates operated on logical qubits q_1, q_2 as $y(q_1, q_2) \in \mathbb{N}$.

$$y(q_1, q_2) = G.count(CNOT(q_1, q_2)).^1$$

Denote the cost of a remote CNOT gate between processors v_1 and $v_2 \in V$ as $c(v_1, v_2)$. Based on the graph H, we can compute the distance between any two nodes $v_1, v_2 \in V$,

[1] $list.count(value)$ function returns the number of elements with the specified value in the list.

i.e., the number of hops of the shortest path between v_1, v_2 on graph H, denoted as $dist(v_1, v_2)$. Then, according to Section IV-C, we can get if $v_1 \neq v_2$, i.e., $dist(v_1, v_2) \geq 1$,

$$c(v_1, v_2) = C_{epr} \cdot dist(v_1, v_2) + C_{bsm} \cdot (dist(v_1, v_2) - 1) + C_{rcn}. \quad (3)$$

And it is trivial to see $c(v, v) = 0, \forall v \in V$.

Thus, the total cost of all remote CNOT gates after qubit allocation, i.e., the communication overhead, is

$$\mathbb{E} = \sum_{q_1, q_2 \in Q} \sum_{v_1, v_2 \in V} y(q_1, q_2) \cdot c(v_1, v_2) \cdot x_{q_1, v_1} \cdot x_{q_1, v_2}. \quad (4)$$

In all, the QA-DQC problem can be formulated as the below ILP.

$$\begin{aligned} \min \quad & \sum_{q_1, q_2 \in Q} \sum_{v_1, v_2 \in V} y(q_1, q_2) \cdot c(v_1, v_2) \cdot x_{q_1, v_1} \cdot x_{q_1, v_2} \\ \text{s.t.} \quad & \sum_{v \in V} x_{q,v} = 1, \quad \forall q \in Q, \\ & \sum_{q \in Q} x_{q,v} \leq n_v, \quad \forall v \in V. \end{aligned}$$

B. Proof of NP-hardness

This section shows that QA-DQC is NP-hard through a reduction from the Quadratic Assignment Problem (QAP). It means QA-DQC cannot be solved in deterministic polynomial time unless $P = NP$.

Theorem 1. *QA-DQC is NP-hard.*

Proof. As for any QAP problem with facility set P, location set L, a weight function $w : P \times P \longrightarrow R$ and a distance function $d : L \times L \longrightarrow R$, we can generate a QA-DQC problem in polynomial time by setting $Q = P, V = L, y(\cdot) = w(\cdot), c(\cdot) = d(\cdot), n_v = 1(\forall v \in V)$. Here, the generated QA-DQC problem and the given QAP are exactly the same, and thus obviously, they are equivalent.

In all, QAP $\leq_P$ QA-DQC. Since QAP is NP-hard, QA-DQC is also NP-hard. □

C. Problem Complexity and Analysis

Theorem 2. *Unless $P = NP$, there is no polynomial-time n^a-approximation algorithm for QA-DQC, for any $a < 1$. Here n is the number of processors in the quantum network.*

Proof. Hassin et al. [31] has proved there is no polynomial-time n^a-approximation algorithm for the QAP when the graph of locations is a tree, for any $a < 1$, unless $P = NP$. Here n is the number of location nodes in the graph. Below we will prove by contradiction based on this known conclusion.

Assume there is an polynomial-time n^a-approximation algorithm, noted as A_1, for any QA-DQC for some $a < 1$. Here n is the number of processors in the quantum network. In the proof of Thm. 1, we have proved for any problem $p \in$ QAP, there exist a $p' \in$ QA-DQC, such that $p' = p$. Denote the QAP when the graph of locations is a tree as QAP$|_{tree}$. Thus, for any problem $p \in$ QAP$|_{tree} \subseteq$ QAP, there exist a $p' \in$ QA-DQC, such that $p' = p$. Since $p' \in$ QA-DQC, A_1 is an polynomial-time n^a-approximation algorithm for p'. So, it is also an polynomial-time n^a-approximation algorithm for p, for any any problem $p \in$ QAP$|_{tree}$. In all, A_1 is a polynomial-time n^a-approximation algorithm for the QAP when the graph of locations is a tree, which contradicts the conclusion proved by Hassin et al. By contradiction, we have proved our theorem. □

As we said before, the compilation of quantum circuits in distributed quantum computing demands extremely accurate compiler optimization. Thus, approximation algorithms with approximation ratios larger than or equal to n are unacceptable. Pursuing more accurate optimization, we resort to a meta-heuristic algorithm called Simulated Annealing.

VI. Meta-heuristic Algorithm For QA-DQC

A. The framework for MHSA

Simulated Annealing algorithm (SA) [32] simulates the fast heating and slow cooling of metal so that a uniform crystalline state can be achieved to guide the search for an optimal solution to an optimization problem. In general, it is divided into two stages. In the beginning, the solution moves relatively freely in the solution set, where a downhill move is always accepted, while an uphill move can also be accepted by a probability defined by Boltzmann's law. Then the probability of accepting an uphill move will attenuate as time goes on in the slow cooling process. After a certain point, almost no uphill move is accepted, which is considered the second stage. But the problem is that the SA algorithm determines the next move by random selection from the neighborhood of the current solution. Thus, the process of searching for a downhill move is inefficient in this stage.

Inspired by work [33], we plan to design a heuristic Local Search algorithm (NS) developed for QA-QDC and use it to replace the inefficient second stage of SA, generating a hybrid simulated annealing algorithm. Specifically, at the beginning of the cooling process, we first capitalize on the SA algorithm until a stopping point: We stop SA when the random move from the current solution has been continuously rejected by fixed times, i.e., the algorithm has stuck in the current solution for a certain number of consecutive trials. It reflects SA goes into the insufficient second process. Thus, we exchange to the heuristic NS algorithm that can quickly converge to a local optimum of the current solution area.

What's more, it should be noted that, in the state-of-the-art SA algorithms, the temperature changes periodically rather than decreases monotonically [34], which implies a sequence of fast heating and slow cooling is considered instead of straight-forward annealing. Thus, to further improve algorithm performance, we exploit periodic cooling, also called the multistage annealing process.

In general, we design a multistage hybrid simulated annealing algorithm. It starts from a randomly generated solution and first resorts to NS for a local optimum. Then we initialize the temperature (heating process) and call the SA algorithm (cooling process) until a stopping point. Next, employ the NS algorithm to let the current solution fast converge to a local

optimum. Afterward, repeat the heating, cooling, and local search processes a fixed number of times noted as n_{stage}.

Below we will first introduce the designed NS algorithm for the QA-DQC problem.

B. Local Neighborhood Search Algorithm

We plan to design the local search algorithm based on the neighborhood. It is a kind of gradient descent approach. Specifically, we always select a move that maximizes the decrease of the overhead.

We first define a move and the neighborhood for a solution of QA-DQC. In QAP, the popularly-used solution move is realized by a location swap between two facilities. However, QA-DQC is a bit more complicated than QAP. The main difference is that multiple logical qubits may be assigned to the same quantum processor. The maximal number of logical qubits that can be assigned to a quantum processor is limited by the processor's capacity, i.e., the number of computation qubits in this processor.

Like QAP, a qubit swap can help make a solution move in QA-DQC, but the swap must be limited between two logical qubits assigned to different quantum processors. But it is not enough. Only swaps can not produce all solution moves in QA-DQC. There is another kind of essential move, which is realized by moving a certain logical qubit to a different quantum processor with idle computation qubits.

For simplification, we unite these two kinds of solution moves by adding some pseudo logical qubits to occupy all idle computation qubits in all processors of the quantum cluster. By doing so, all solution moves can be realized by a qubit swap between a real logical qubit and another real logical qubit or a pseudo logical qubit. Specifically, moving a certain logical qubit of processor V_p to a different quantum processor V_q with idle computation qubits can be considered as a swap between the logical qubit of processor V_p and a pseudo logical qubit of processor V_q.

In all, we redefine the solution $f_v(q) : Q \longrightarrow V$, by extending its domain Q to $Q'(Q \subseteq Q')$ so that each processor node $v \in V$ has exactly n_v different $q \in Q'$ mapped to it. The physical meaning is adding some pseudo logical qubits to occupy all idle computation qubits in all processors of the quantum cluster. Then a move is exactly a swap between two (real or pseudo) logical qubits that are assigned to a different quantum processor. And the neighborhood of a solution $f_v(q)$ is defined by $\mathcal{N}(f_v) = \{f'_v | f'_v(q_1) = f_v(q_2), f'_v(q_2) = f_v(q_1), \forall q_1, q_2 \in Q', f_v(q_1) \neq f_v(q_2); f'_v(q) = f_v(q), \forall q \in Q', q \neq q_1, q_2.\}$. Our local neighborhood search algorithm works as follows: Find the solution f'_v with minimal overhead among the neighborhood of the current solution $\mathcal{N}(f_v)$. If the overhead of the newfound solution f'_v is lower than that of the current solution f_v, move to the found solution f'_v, mark it as the current solution, i.e., $f_v = f'_v$, and repeat the above steps. Otherwise, a local optimum has been found, and the local neighborhood search ends.

C. Multistage Hybrid Simulated Annealing Algorithm

a) ***Principle of the Simulated Annealing****:* The principle of the Simulated Annealing is as follows: Given a solution f_v, randomly select a neighbouring solution $f'_v \in \{_{\sqsubseteq}$ and compute the difference between their corresponding overhead, $\Delta\mathbb{E} = \mathbb{E}(f'_v) - \mathbb{E}(f_v)$. (Recall that $\mathbb{E}(\cdot)$ here is the optimization objective function defined in Eq. 4.) If the solution overhead is reduced, i.e., $\Delta\mathbb{E} < 0$, then replace the current solution f_v by the new one f'_v, i.e., perform a move, which is a downhill move, and use resulting configuration as the starting point for the next trial. If $\Delta\mathbb{E} \geq 0$, which means the move will increase the solution overhead, then accept such an uphill move with probability.

$$P(\Delta\mathbb{E}) = e^{-\frac{\Delta\mathbb{E}}{t}}, \tag{5}$$

where t is the current temperature value. Note that such accepting probability computation follows Boltzmann's law. Regarding the probabilistic acceptance of uphill move, it is achieved by generating a random number in $[0, 1]$ and comparing it against the threshold $P(\Delta\mathbb{E})$.

This procedure is repeated until the termination criterion is satisfied. And finally, the "best-so-far" (BSF) solution, rather than the current solution, is regarded as the result of the algorithm.

b) ***Cooling Schedule****:* The cooling schedule consists of (1) an initial value of the temperature, (2) an updating function for changing the temperature, and (3) an equilibrium test.

The "behavior" of the simulated annealing algorithm depends on the temperature. Perhaps the most important thing is to select a suitable initial temperature t_0. Since in the framework of our MHSA, the employed simulated annealing algorithm always starts from a local optimum. And at the beginning of the slow cooling process, we want it to move relatively freely so that it can escape from the current local optima and move to another possible good solution area. Thus, it asks for a relatively high initial temperature t_0. But a too high initial temperature may lead to some meaningless move and unprofitable time cost. So choosing a relatively high, but not too high, initial temperature is essential for the performance of our MHSA.

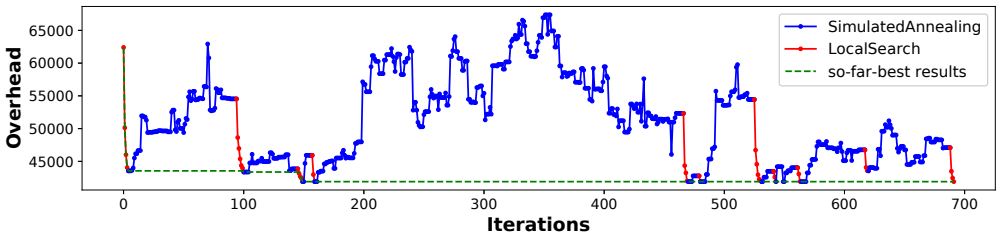
The temperature updating function we choose is

$$t_{k+1} = \alpha \cdot t_k, \quad \forall k \geq 0, \ (t_0 = const, \alpha < 1) \tag{6}$$

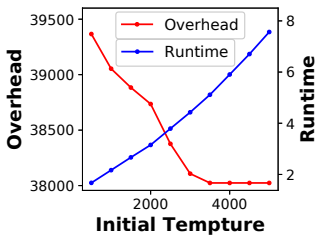
which is a popularly used schedule [32] in SA.

Besides, Kirkpatrick et al. proposed that, at each temperature, the cooling schedule must allow the simulation to proceed long enough for the process to reach a steady state – equilibrium. A simple but typically used equilibrium test is that the temperature is decreased after a fixed number of trials, noted as n_{eqlb}.

c) ***Termination Criterion****:* According to the framework of MHSA, the termination criterion is that the move from the current solution has been continuously rejected by a fixed number of times, i.e., the algorithm has stuck in the current solution for a certain number of consecutive trials, noted as n_{stuck}.



(a) The heating, cooling and local search process of our MHSA with 10 stages ($n_{stage} = 10$). The red line is the local search part of MHSA, while the blue line is the cooling process of SA in MHSA. At the point where the red line becomes blue, there is an instant heating process. The green line exhibits the so-far-best results.



(b) The initial temperature selection process

Fig. 4: Performance evaluation of MHSA. (The tested quantum circuit is *ham15*, while the quantum cluster is star topology.)

VII. PERFORMANCE EVALUATION

In this section, we perform extensive simulations to evaluate the performance of the MHSA and compare it with the benchmarks.

A. Simulation Setting

*a) **Tested Quantum Circuits**:* In our simulations, we test 8 different quantum circuits listed in Table. I. They are functions taken from RevLib [35] and the corresponding compiled .qasm[2] file can be founded in [36].

Name	num. of qubits	num. of CNOTs	initial temp. t_0
410184	14	104	300
clip	14	14772	20000
cm42a	14	771	1100
sao2	14	16864	20000
ham15	15	3858	4000
dc2	15	4131	5000
co14	15	7840	10000
misex1	15	2100	4000

TABLE I: Different tested quantum circuits

*b) **Quantum Cluster and Network Topology**:* We consider all the processors in the quantum cluster have 5 physical qubits, like IBM Yorktown, Vigo, Burlington, etc. Since each quantum link needs to occupy 2 physical qubits work as communication qubits and our tested quantum circuits need 14-15 computation qubits, we implement our simulations on the following 8 different network topologies with 5 nodes and at most 5 links, shown in Fig. 5: **1) star** (4 links, 17 computation qubits), **2) line** (4 links, 17 computation qubits), **3) cross** (4 links, 17 computation qubits), **4) ring** (5 links, 15 computation qubits), **5) quad** (5 links, 15 computation qubits), **6) star-traingl** (5 links, 15 computation qubits), **7) line-triangl** (5 links, 15 computation qubits), **8), cross-triangl** (5 links, 15 computation qubits).

*c) **Parameters Setting**:* We set the EPR pair generation cost $C_{epr} = 7$, the cost of a Bell-state measurement circuit $C_{bsm} = 3$ and the cost of a remote CNOT circuit $C_{rcn} = 5$. Besides, in MHSA, we set the cooling parameter $\alpha = 0.9$, the fixed number of trials for equilibrium test $n_{eqlb} = 5$, the stop point $n_{stuck} = 5$, the number of repeated heating and cooling stages $n_{stage} = 50$.

[2]QASM is a quantum programming language

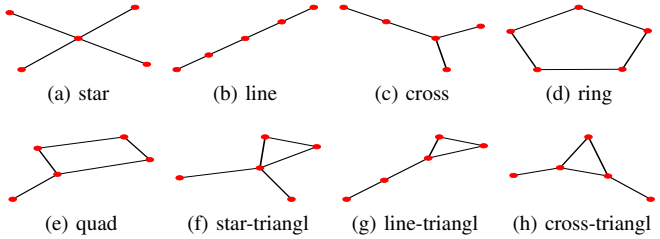


Fig. 5: Different quantum cluster / network topologies

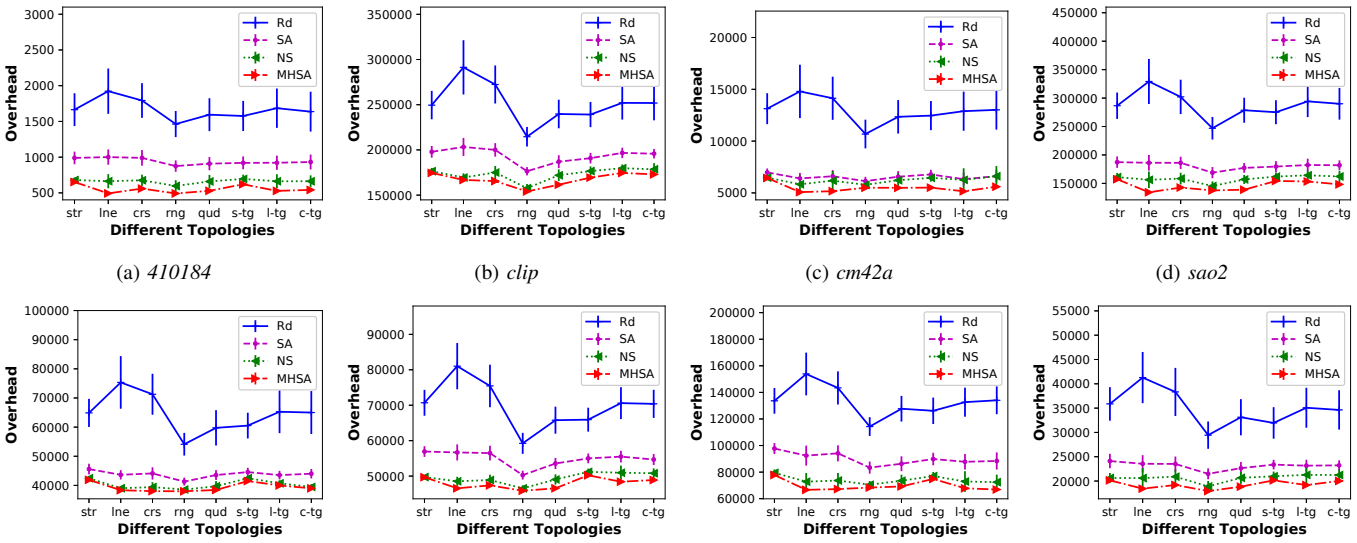
*d) **Benchmarks**:* We will compare MHSA with the following 3 solutions for QA-DQC: **1) RanDom feasible solution (Rd)**: the algorithm returns a random feasible solution for QA-DQC; **2) isolated Neighborhood Search algorithm (NS)**: the algorithm is exactly the local neighborhood search part in our MHSA introduced in Section VI-B; **3) isolated Simulated Annealing algorithm (SA)**: the algorithm is a popularly-used meta-heuristic algorithm.

Note that since the randomly produced initial solution will affect the performance of NS, SA, and MHSA to some extent, we always use the average value of 100 groups of simulations to show moderate cases in all the plots below except Fig. 4a.

B. Performance Evaluation of MHSA

We first use a tested quantum circuit called *ham15* and the topology of the quantum cluster called star to evaluate the performance of MHSA in Fig. 4.

Fig. 4a shows the 10 stages of the heating, cooling, and local search process of our MHSA, where the x-axis value is the index of each iteration, or to say, each solution move, and the y-axis value is the overhead, i.e., the total cost of all remote CNOT operations. In the plot, the red line is the local search part of MHSA, while the blue line is the cooling process of SA in MHSA. At the point where the red line becomes blue, which is the beginning of a stage of the heating, cooling, and local search process, there is an instant heating process. The green line exhibits the so-far-best results. In this plot, we can see that after each red-to-blue point, which is also the local optimal point, there are some uphill moves that help escape the local optimal point. Afterward, as the cooling process (the blue line), more and more downhill moves appear. To some stuck point (the blue-to-red point), the MHSA turns to employ the local neighborhood search algorithm (the red line) to realize a fast drop to the local optimal solution (the next red-to-blue point).



(a) *410184* (b) *clip* (c) *cm42a* (d) *sao2*

(e) *ham15* (f) *dc2* (g) *co14* (h) *misex1*

Fig. 6: Performance Comparisons of MHSA and three benchmarks when allocating qubits of 8 different quantum circuits to the processors of quantum clusters with 8 different topologies

The whole process realizes our expectations when designing MHSA in Section VI-A.

Fig. 4b exhibits different initial temperature selections and the corresponding overhead and run-time, which reveals that as the selected initial temperature increases, the overhead first decreases and then becomes steady while the run-time monotonically increases. It is consistent with our analysis in Section VI-C(b): relatively high initial temperature helps escape from the current local optima to exploit better solutions, but the too high initial temperature may cause unprofitable time cost without performance improvement. In Fig. 4b, we can see the cost-efficient initial temperature for quantum circuit *ham15* and a star-topology quantum cluster is 3500. Considering other cluster topologies, the final selected initial temperatures for each quantum circuit are shown in Table I.

C. Performance Comparison on Different Topologies

Fig. 6 lists the performance comparison of MHSA and three benchmarks when allocating qubits of 8 different quantum circuits to the processors of quantum clusters with 8 different topologies, where each subplot is the case of qubit allocation of each quantum circuit on 8 different cluster topologies. As we can see from Fig. 6, MHSA always maintains better performance than the benchmarks regardless of the quantum circuits and cluster topologies. Besides, the standard variances of all overhead results of MHSA in all subplots of Fig. 6 are near zero, which implies the results of MHSA are close to the optimal outcomes. It is because local optima always depend on the randomly produced initial solution to some extent, which will cause large variance. In 100 groups of repeated simulations, only global optima will always maintain the same, producing zero variance. It can also be reflected by the phenomenon that the results of benchmarks with worse performance have larger standard variance in the plots.

Additionally, the best suitable cluster topology, i.e., the topology with the lowest overhead by MHSA, for different quantum circuits are different. As for the quantum circuits *410184, misex1, dc2, clip, ham15*, the best cluster topology is ring, while the best cluster topology for the other quantum circuits, i.e., *sao2, cm42a, co14* is line. Such results shed some light on the design of distributed quantum networks, i.e., how to connect quantum processors to produce clusters.

VIII. CONCLUSION AND FUTURE WORK

With the development of quantum communication, distributed quantum computing can produce more computational power, where different quantum processors can communicate and cooperate in executing more complicated computational tasks that a single quantum processor cannot handle. But it also brings challenges to the special compiler design, i.e., how to allocate logical qubits to different quantum processors. This paper builds a model for a general qubit allocation problem for distributed quantum computing. We first prove its NP-hardness. Moreover, we further prove there is no polynomial-time n^a-approximation algorithm to solve it for any $a < 1$ unless $P = NP$, where n is the number of processors in the quantum network. To deal with this issue, we design a multistage hybrid simulated annealing algorithm, which combines a meta-heuristic algorithm called simulated annealing and a designed heuristic local search algorithm for QA-DQC. Lastly, we perform extensive simulations on various real quantum circuits and different network topologies, which demonstrates that the proposed MHSA always outperforms the baselines and is close to the optimal results.

IX. ACKNOWLEDGE

This research work was supported in part by the National Science Foundation under grant number CNS-2231040.

REFERENCES

[1] A. Politi, J. C. Matthews, and J. L. O'brien, "Shor's quantum factoring algorithm on a photonic chip," *Science*, vol. 325, no. 5945, pp. 1221, 2009.

[2] E. Gibney, "The quantum gold rush," *Nature*, vol. 574, no. 7776, pp. 22–24, 2019.

[3] D. Cuomo, M. Caleffi, and A. S. Cacciapuoti, "Towards a distributed quantum computing ecosystem," *IET Quantum Communication*, vol. 1, no. 1, pp. 3–8, 2020.

[4] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, et al, " Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.

[5] J. Chow, O. Dial, and J. Gambetta, " IBM Quantum breaks the 100-qubit processor barrier," *IBM Research Blog*, 2021.

[6] Y. Zhou, E. M. Stoudenmire, and X. Waintal, "What limits the simulation of quantum computers?," *Physical Review X*, vol. 10, no. 4, pp. 041038, 2020.

[7] W. Kozlowski, and S. Wehner, "Towards large-scale quantum networks," *Proceedings of the Sixth Annual ACM International Conference on Nanoscale Computing and Communication*, pp. 1–7, 2019.

[8] A. S. Cacciapuoti, M. Caleffi, F. Tafuri, F. S. Cataliotti, S. Gherardini, and G. Bianchi, "Quantum internet: networking challenges in distributed quantum computing," *IEEE Network*, vol. 34, no. 1, pp. 137–143, 2019.

[9] M.Y. Siraichi, V.F.D. Santos, C. Collange, and F.M.Q. Pereira, "Qubit allocation,"*Proceedings of the 2018 International Symposium on Code Generation and Optimization*, pp. 113–125, 2018.

[10] A. Ash-Saki, M. Alam, and S. Ghosh, "QURE: Qubit re-allocation in noisy intermediate-scale quantum computers,"*Proceedings of the 56th Annual Design Automation Conference*, pp. 1–6, 2019.

[11] P. Benioff, "The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines,"*Journal of statistical physics*, vol. 22, no. 5, pp. 563–591, 1980.

[12] O. Regev, " Quantum computation and lattice problems,"*SIAM Journal on Computing*, vol. 33, no. 3, pp. 738–760, 2004.

[13] R. Van Meter, S. J. Devitt, "The path to scalable distributed quantum computing,"*Computer*, vo. 49, no. 9, pp. 31-–42, 2016.

[14] M. Zomorodi-Moghadam, M. Houshmand, and M. Houshmand, "Optimizing teleportation cost in distributed quantum circuits,"*International Journal of Theoretical Physics*, vol. 57, no. 3, pp. 848–861, 2018.

[15] P. Andres-Martinez, and C. Heunen, "Automated distribution of quantum circuits via hypergraph partitioning,"*Physical Review A*, vol. 100, no. 3, pp. 032308, 2019.

[16] Z. Davarzani, M. Zomorodi-Moghadam, M. Houshmand, and M. Nouribaygi, "A dynamic programming approach for distributing quantum circuits by bipartite graphs,"*Quantum Information Processing*, vol. 19, no. 10, pp. 1–18, 2020.

[17] Y. Akhremtsev, T. Heuer, P. Sanders, and S. Schlag, "Engineering a direct k-way hypergraph partitioning algorithm.,"*Proceedings of the Nineteenth Workshop on Algorithm Engineering and Experiments (ALENEX). Society for Industrial and Applied Mathematics*, pp. 28–42, 2017.

[18] A. Yimsiriwattana, and S.J. Lomonaco Jr, "Distributed quantum computing: A distributed Shor algorithm,"*Quantum Information and Computation II*, vol. 5436, pp. 360–372, SPIE, 2004.

[19] R. Beals, S. Brierley, O. Gray, A. W. Harrow, S. Kutin, N. Linden, D. Shepherd, and M. Stather, "Efficient distributed quantum computing,"*Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 469, no. 2153, pp 20120686, 2013.

[20] D. Ferrari, A.S. Cacciapuoti, M. Amoretti, and M. Caleffi, "Compiler design for distributed quantum computing,"*IEEE Transactions on Quantum Engineering*, 2020.

[21] A. Barenco, C.H. Bennett, R. Cleve, D.P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J.A. Smolin, and H. Weinfurter, "Elementary gates for quantum computation," *Physical review A*, vol. 52, no. 5, pp. 3457, 1995.

[22] M.H. Devoret, A. Wallraff, and J.M. Martinis, "Superconducting qubits: A short review," *arXiv preprint cond-mat/0411174*, 2004.

[23] J. Koch, M.Y. Terri, J. Gambetta, A.A. Houck, D.I. Schuster, J. Majer, A. Blais, M.H. Devoret, S.M. Girvin, and R.J. Schoelkopf, "Charge-insensitive qubit design derived from the Cooper pair box," *Physical Review A*, vol. 76, no. 4, pp. 042319, 2007.

[24] J.M. Gambetta, J.M. Chow, and M. Steffen, "Building logical qubits in a superconducting quantum computing system," *npj Quantum Information*, vol. 3, no. 1, pp. 1–7, 2017.

[25] Y.F. Huang, X.F. Ren, Y.S. Zhang, L.M. Duan, and G.C. Guo, "Experimental teleportation of a quantum controlled-NOT gate,"*Physical review letters*, vol. 93, no. 24, pp. 240501, 2004.

[26] G.F. Dang, and H. Fan, "Remote controlled-NOT gate of d-dimension,"*arXiv preprint arXiv:0711.3714*, 2007.

[27] J. W. Pan, D. Bouwmeester, H. Weinfurter, and A. Zeilinger, "Experimental entanglement swapping: entangling photons that never interacted,"*Physical review letters*, vol. 80, no. 18, pp. 3891, 1998.

[28] A. M. Goebel, C. Wagenknecht, Q. Zhang, Y.A. Chen, K. Chen, J. Schmiedmayer, and J. W. Pan, "Multistage entanglement swapping,"*Physical Review Letters*, vol. 101, no. 8, pp. 080403, 2008.

[29] D. A. Lidar, and T. A. Brun, "Quantum error correction,"*Cambridge university press*, 2013.

[30] M. Mohseni, P. Read, H. Neven, S. Boixo, V. Denchev, R. Babbush, A. Fowler, V. Smelyanskiy, and J. Martinis, "Commercialize quantum technologies in five years,"*Nature*, vol. 543, no. 7644, pp. 171–174, 2017.

[31] R. Hassin, A. Levin, and M. Sviridenko, "Approximating the minimum quadratic assignment problems,"*ACM Transactions on Algorithms (TALG)*, vol. 6, np. 1, pp. 1–10, 2009.

[32] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, "Optimization by simulated annealing,"*science* vol. 220, no. 4598, pp. 671–680, 1983.

[33] O.C. Martin, and S.W. Otto, "Combining simulated annealing with local search heuristics,"*Annals of operations research*, vol. 63, no. 1, pp. 57–75, 1996.

[34] A. Misevičius, "A modified simulated annealing algorithm for the quadratic assignment problem,"*Informatica*, vol. 14, no. 4, pp. 497–514, 2003.

[35] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler, "RevLib: An online resource for reversible functions and reversible circuits,"*38th International Symposium on Multiple Valued Logic (ismvl)*, pp. 220–225, 2008. RevLib is available at http://www.revlib.org.

[36] https://github.com/cda-tum/qmap/tree/main/examples.