Journal of
Quantum Computing

Tech Science Press

Doi:10.32604/jqc.2025.061275

ARTICLE

A Genetic Approach to Minimising Gate and Qubit Teleportations for Multi-Processor Quantum Circuit Distribution

Oliver Crampton[1,*], Panagiotis Promponas[1,2], Richard Chen[1], Paul Polakos[1], Leandros Tassiulas[2] and Louis Samuel[1]

[1]Cisco Systems UK, Bedfont Lakes, London, TW14 8HA, UK
[2]Department of Electrical Engineering, Yale University, New Haven, CT 06511, USA

*Corresponding Author: Oliver Crampton. Email: omc2000@hw.ac.uk

Received: 21 November 2024; Accepted: 20 February 2025; Published: 21 March 2025

ABSTRACT: Distributed Quantum Computing (DQC) provides a means for scaling available quantum computation by interconnecting multiple quantum processor units (QPUs). A key challenge in this domain is efficiently allocating logical qubits from quantum circuits to the physical qubits within QPUs, a task known to be NP-hard. Traditional approaches, primarily focused on graph partitioning strategies, have sought to reduce the number of required Bell pairs for executing non-local CNOT operations, a form of gate teleportation. However, these methods have limitations in terms of efficiency and scalability. Addressing this, our work jointly considers gate and qubit teleportations introducing a novel meta-heuristic algorithm to minimise the network cost of executing a quantum circuit. By allowing dynamic reallocation of qubits along with gate teleportations during circuit execution, our method significantly enhances the overall efficacy and potential scalability of DQC frameworks. In our numerical analysis, we demonstrate that integrating qubit teleportations into our genetic algorithm for optimizing circuit blocking reduces the required resources, specifically the number of EPR pairs, compared to traditional graph partitioning methods. Our results, derived from both benchmark and randomly generated circuits, show that as circuit complexity increases—demanding more qubit teleportations—our approach effectively optimises these teleportations throughout the execution, thereby enhancing performance through strategic circuit partitioning. This is a step forward in the pursuit of a global quantum compiler which will ultimately enable the efficient use of a 'quantum data center' in the future.

KEYWORDS: Distributed quantum computing; optimisation; teleportation; heuristic

1 Introduction

Quantum computers can, in principle, perform tasks that have previously been impossible or highly inefficient on classical computers [1], such as factoring large numbers using Shor's algorithm [2,3] or simulating quantum systems [4,5]. However, the scaling of monolithic quantum processors towards doing useful, error-free computations is difficult to achieve [6]. To address this, companies such as IBM are exploring inter-connected distributed quantum processor units (QPUs), which require quantum networking to execute complex circuits at scale. Similar to classical distributed computing, the execution of a quantum circuit may be performed across many smaller quantum computers. In this scenario, a quantum network is needed to perform the necessary operations between each QPU.

Quantum circuits are a visual way to represent the temporal order of single or multi-qubit gates, to perform a designed algorithm. Quantum gates are unitary operations that operate on a logical qubit state

$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$. A logical qubit is the unit of information used in a quantum computation and may be constructed from many physical qubits inside a QPU to perform error correction. A more detailed review of quantum information and quantum computing can be found in the seminal text by Nielsen et al. [7]. In Distributed Quantum Computing (DQC) there are three important types of operations, single-qubit gates (e.g., Pauli rotation, Hadamard), local CNOT (cx) (control and target qubits within the same processor), and non-local CNOT (also called telegate). In the latter, a CNOT operation should be executed between qubits that are stored in different QPUs. The Hadamard gate, Pauli gates, and CNOT operations form a universal set for quantum computation [8], meaning that any arbitrary unitary transformation of a quantum state can be expressed by only these three gates [9], henceforth we will assume that all circuits have been decomposed into this universal set of operations. The CNOT gate is a two-qubit operator which requires the control of one qubit by another and is given by the following matrix:

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{1}$$

To execute a distributed algorithm, the logical qubits must be mapped to physical qubits in the QPUs and a means for performing operations between non-local qubits must be established. Two types of non-local operations can occur, qubit or gate teleportation. Qubit teleportation is the process of transferring a logical qubit state from one QPU to another via the use of a Bell state (EPR pair) [10] and some classical communication. Gate teleportation, on the other hand, does not move the qubit state but allows one qubit to control the operation on another, distant qubit. The circuits required to perform both qubit teleportation and a teleported controlled operation between two distant logical qubits [11] are shown in Fig. 1. Both operations make use of Bell states which need to be distributed by the network (one-half to each QPU). This is a costly procedure and hence there is a need for minimising 2 the teleportations to reduce the load on the network and to maximise the likelihood of successfully executing a quantum circuit before qubits decohere.

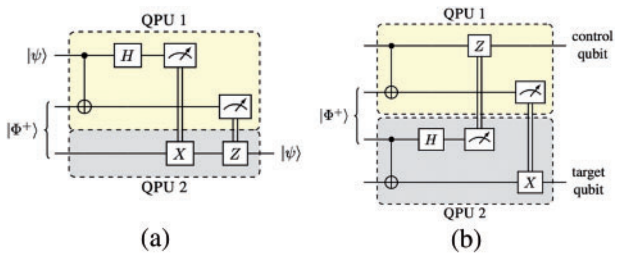


Figure 1: Circuits that implement (a) a teleportation operation of a state $|\psi\rangle$ from QPU 1 to QPU 2, and (b) a teleportation of a CNOT operation between a control qubit and target qubit that are stored in different QPUs. Both operations require a Bell state $|\Phi+\rangle$ as well as the transmission of classical bits that correspond to the outcome of qubit measurements

When executing a quantum circuit on a single QPU, it's crucial for the compiler to dynamically map logical qubits to neighboring physical positions. This mapping allows gate operations by enabling direct interactions. Numerous studies have addressed the challenges and solutions related to quantum circuit compilation (e.g., [12–14]) in the case of a single QPU. Our work, however, assumes a fully connected architecture for QPUs, where each qubit can directly interact with any other. A similar assumption would be the existence of an efficient compiler that handles the qubit mapping inside each QPU separately. Such assumptions allow us to abstract away the constraints of the compilation problem, focusing instead on optimising the network operations necessary for distributed quantum computation. Assuming only gate teleportations as a means towards DQC, previous works have used various heuristic methods to minimise solely the number of non-local (controlled) operations within a circuit [15–19].

Although previous works have primarily considered the minimisation of gate teleportation, this work sets out to jointly consider gate and qubit teleportations as enablement of DQC. Thus, the network cost is associated with the Bell pairs requested from both teleportation operations. Recently the authors in [20] introduced a graph partitioning framework that incorporates qubit teleportations into network cost calculations. While their work also recognizes the importance of qubit teleportations in DQC, it differs from ours in its constraint of equalizing operation counts across QPUs. Our study, in contrast, views the generation of Bell pairs for network operations as the primary limiting factor, thereby allowing for more flexible QPU operation allocations. Recently, reference [21] employs Quadratic Unconstrained Binary Optimisation to minimise the network cost assuming only qubit teleportations. In the latter work, the authors divide a quantum circuit into predetermined slices such that each such slice can be run without the need for gate teleportations. Finally, in [22], the authors employ a window-based partitioning of a quantum circuit considering also qubit teleportations. Nevertheless, the optimisation over the latter is realized through a tuning parameter that determines how "hard" should be for a qubit to migrate to a different QPU. In contrast, in our work, we optimise the slicing of the circuit by allowing gate teleportations within each slice and qubit teleportations across slices to facilitate distributed quantum computation.

Specifically, this paper addresses the problem of modeling and minimising the network cost of executing a quantum circuit into a DQC framework. Allowing both gate and qubit teleportations, we dynamically allocate the logical qubits to physical qubits into the quantum processors to execute a quantum algorithm distributedly. Since both teleportation operations require a Bell pair our goal is to minimise the number of teleportations needed to complete the execution. For this purpose, we propose a novel meta-heuristic called Optimised Distributed Quantum Circuit Execution via Meta-Heuristic Approach (ODQC-MHA) that uses a genetic algorithm. Our method significantly enhances the overall efficacy and potential scalability of the DQC framework by dynamically allocating the qubits across distributed QPUs.

The rest of the paper is organised as follows, in Section 2 we introduce the logical qubit allocation problem for static assignment within a monolithic QPU as well as a straightforward partitioning heuristic. Section 3 introduces qubit teleportations to the qubit mapping problem and we describe our meta-heuristic for solving this problem approximately. Section 4 shows the results of the performance of our metaheuristic against benchmark circuits and randomly generated circuits. Finally, Section 5 concludes the paper.

2 Qubit Allocation to Minimise Gate Teleportations

Focusing only on gate teleportations as the means towards DQC, one way to decrease the network cost is to leverage graph partitioning algorithms [23–25] in an appropriately generated graph. In this section, we describe such process for the case of two and multiple QPUs (Sections 2.2 and 2.3, respectively).

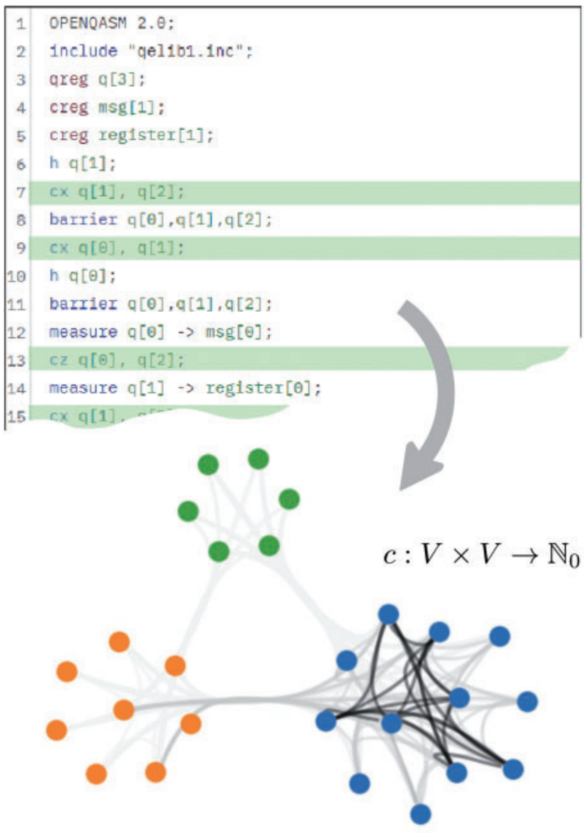
In this study, we operate under the assumption that there is complete connectivity both between and within the QPUs. This means we overlook any compilation within a QPU and presume that the compiler handles the required swaps in a non-fully connected QPU to ensure adjacent qubits. Research has been done on simulating circuits on realistic, constrained processor architectures by minimising the number of swaps required to execute controlled operations [26]. In practice constraints on the connectivity within a processor are likely to exist, however, one would need a global compiler to be able to maximise the ability to execute a circuit on NISQ devices.

2.1 Minimising Gate Teleportation: Model & Problem Description

In the graph representation of a circuit denoted as $G = (V, E)$, V represents a set of n qubits and E defines connections between qubits. The edge-weight function

$$c: V \times V \to \mathbb{N}_0 \tag{2}$$

where $\mathbb{N}_0$ denotes the set of natural numbers including zero represents the frequency of controlled operations between qubits u and v. Therefore, $c(u, v) = 0$ indicates the absence of an edge and thus of a CNOT gate between u and v. The cost of a partition $cost(V_1, V_2)$ is defined as the sum of all weights $c(u, v)$ where $u \in V1$ and $v \in V2$ belong to different partitions. The aim is to find k partitions of the graph each of, at most, size $v = n/k$ such that the capacity of the edges between partitions is minimised, thus reducing the number non-local controlled operations, and the number of Bell pairs required. The constraint of almost equal partitions is to minimise the maximum number of physical qubits needed from a QPU. Minimising the number of non-local operations is crucial because entanglement is a costly resource and distributing it into a network of QPUs requires extra time steps and is error-prone [27]. A schematic showing the graph partitioning method is shown in Fig. 2.



```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[3];
creg msg[1];
creg register[1];
h q[1];
cx q[1], q[2];
barrier q[0],q[1],q[2];
cx q[0], q[1];
h q[0];
barrier q[0],q[1],q[2];
measure q[0] -> msg[0];
cz q[0], q[2];
measure q[1] -> register[0];
cx q[1],
```

Figure 2: Example of initial qubit allocation process via graph partitioning to minimise non-local operations between 3 processors of sizes: (6, 8, 12). The edge weight is signified by the line colour, darker lines represent higher number of non-local operations

2.2 K-L Algorithm for 2 QPUs

One commonly used heuristic method for bi-partitioning a weighted graph is the Kernighan-Lin algorithm [23]. The K-L algorithm works by taking the weighted graph $G = (V, E)$ and c as the edge-weight function. By swapping pairs of vertices $u_i \in V_1$ and $v_i \in V_2$ with maximum cost improvement, the swapped pair are locked in place and the same process is done with another pair until all vertices are locked. The best configuration is chosen and the algorithm is run again until a close to optimal configuration is found. The downside of the K-L algorithm is that it can only be used for bi-partitioning of a graph. In the next section we propose a similarly simple heuristic for partitioning a graph, into any size partitions k. Such extension enables the division of a quantum circuit's logical qubits across multiple QPUs, beyond just two, when available. For all cases where the K-L algorithm is used in this study, sufficiently high numbers of iterations are used to ensure that the algorithm is allowed to "converge".

2.3 Greedy Partitioning Algorithm in the Case of Multiple QPUs

In this section we describe Greedy Partitioning Algorithm in the case of multiple QPUs (GPA), a straightforward and practical heuristic approach for distributing qubits in circuits with varying numbers of qubits and depth, across any quantity and scale of QPUs. The heuristic works by always contracting the largest weighted edge to build supernodes—respresenting QPUs—one by one. Once a supernode (QPU) has been filled to capacity, none of the nodes within can be swapped later in the algorithm. The largest weighted edge that is adjacent to the supernode is always contracted at each step of the heuristic, with no look-ahead. This method is computationally inexpensive and so can be used as a quick heuristic for qubit allocation within our proposed meta-heuristic (proposed in Section 3). The pseudo-code for GPA is shown in Algorithm 1. This heuristic is performed once to produce a good allocation of qubits to processors where the number of interactions between each processor is reduced. The algorithm is implemented in python and utilises the NetworkX framework to do the edge contractions. Here, an edge contraction is the process of producing a graph in which two node v_1 and v_2 are replaced with a single node, v, such that v is adjacent to the union of the nodes to which v_1 and v_2 were originally adjacent, also called 'vertex identification'.

Algorithm 1: Greedy partitioning algorithm (GPA)

```
1  procedure AssignQubitsToQPUs (G, QPU List)
2      Filled ← {}
3      Allocation ← {}
4      QPUOrder ← sort QPU List by capacity, descending
5      for QPU in QPUOrder do
6          Supernode ← {}
7          while Supernode size < QPU capacity and G has unallocated nodes do
8              TargetNode ← select node in G with max weighted edge to Supernode, not in Filled
9              Merge TagetNode into Supernode
10             Update Allocation to include TargetNode → QPU
11         end while
12             Add Supernode to Filled
13     end for
14     return Allocation
15 end procedure
```

3 Minimising Remote Operations: Model & Problem Description

Focusing only on gate teleportations as the means towards DQC, one way to decrease the network cost is to leverage graph partitioning algorithms [23–25] in an appropriately generated graph. In this section, we describe such a process for the case of two and multiple QPUs (Sections 2.2 and 2.3, respectively). In this study, we operate under the assumption that there is complete connectivity both between and within the QPUs. This means we overlook any compilation within a QPU and presume that the compiler handles the required swaps in a non-fully connected QPU to ensure adjacent qubits. Research has been done on simulating circuits on realistic, constrained processor architectures by minimising the number of swaps required to execute controlled operations [26]. In practice constraints on the connectivity within a processor are likely to exist, however, one would need a global compiler to be able to maximise the ability to execute a circuit on NISQ devices.

3.1 Optimised Distributed Quantum Circuit Execution via Meta-Heuristic Approach (ODQC-MHA)—High Level Design

In this section, we introduce the ODQC-MHA framework by describing its high-level design. ODQC-MHA allows the circuit to be analysed in blocks of varying size, using any algorithm for k partitioning a graph within each block. For each block, we attempt to minimise the number of gate teleportations while allowing qubit teleportations between blocks to reallocate the logical qubits when needed. Finding the optimal blocking is challenging because of the many possible combinations, however, this problem is to be solved by our heuristic. Note that each allocation has no information about the previous block's allocation and so a meta-heuristic is required to minimise the gate and qubit teleportations together. The high-level description of the proposed framework is illustrated in Fig. 3. Note that to enhance circuit execution efficiency, qubit teleportations are allowed between blocks. Given the vast search space comprising various circuit partitions and qubit placements, the approach combines any partitioning algorithm for intra-block qubit placements (this can be K-L, GPA, or any other procedure) with a genetic algorithm to jointly consider the qubit teleportations. The proposed genetic algorithm evaluates its utility given a circuit partition based on the placements suggested by the particular partitioning algorithm used, focusing on achieving the optimisation objective of minimising the network cost. Note that this is not a joint optimisation but a meta-heuristic that uses the output of some heuristic (K-L, Greedy algorithm, etc.) to explore the solution space more thoroughly. The problem of graph partitioning is NP-HARD per block hence the blocking model proposed in this paper is hard to solve without a novel heuristic. In the next sections, a genetic algorithm is proposed to approximate an optimal 'blocking' of the circuit to minimise the total number of Bell pairs required for qubit and gate teleportations.

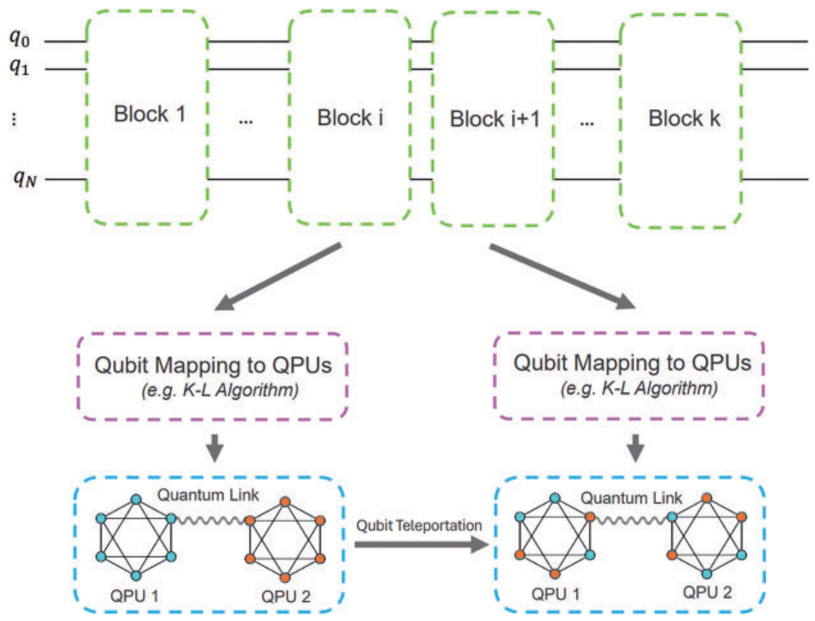


Figure 3: A high level overview diagram of the proposed framework (ODQC-MHA) for blocking/partitioning quantum circuits. The circuit is broken into blocks of arbitrary size (number of layers). Within each block, logical qubit allocation is performed using graph partitioning methods in order to reduce the number of Bell pairs required for nonlocal operations. After this, between each block, qubit teleportations are performed to reallocate the logical qubits according to each blocks allocation

3.2 Optimised Distributed Quantum Circuit Execution via Meta-Heuristic Approach (ODQC-MHA)—Genetic Algorithm

Genetic algorithms mimic natural evolution by evolving solutions to problems through a process of selection, mutation, and crossover [28]. They start with a diverse population of individuals, where each individual's "genotype" encodes a potential solution, and its "phenotype"—its performance or fitness—reflects the solution's effectiveness. Over successive generations, individuals with higher fitness are more likely to pass their genes to the next generation, allowing the algorithm to "naturally select" increasingly effective solutions. In our approach, we utilize a genetic algorithm to optimise the distribution of computational tasks in a quantum computing network, specifically aiming to minimise the requisite number of Bell pairs for efficient quantum communication. The core of our algorithm is defined by a population of candidate solutions, denoted as $P = \{p_1, p_2, \ldots, p_N\}$, where each candidate solution pi represents a potential configuration of dividing the target quantum circuit into distinct blocks.

Each candidate solution $p \in P$ is characterized by its genotype, G_p, which in our model is a sequence of integers $G_p = \{g_1, g_2, \ldots, g_K\}$, where $g_1, \ldots, g_k \in \mathbb{N}$, and K represents a predefined maximum number of blocks for the quantum circuit. Here, gi signifies the depth (i.e., the number of layers) of the circuit within block i of the network. Note that a large value for K increases the search space exponentially allowing for more combinations of circuit blockings to be checked by the algorithm. Intuitively, the configuration of these blocks is subject to a constraint where the sum of all gi values must equal the total number of layers in the quantum circuit. Notably, it is permissible for any gi to be zero, indicating blocks that are empty and thus not contributing to the overall division of the circuit. For instance, a genotype G_p = [12, 42, 64, 38, 203, 0, 34] represents a circuit partitioned into blocks with respective depths of 12, 42, 64, 38, 203, 34. Although in this specific example, we allow up to 7 blocks for the circuit, this specific genotype, G_p, utilizes only 6 of them.

The phenotype, X_p, associated with an individual $p \in P$ with genotype G_p, quantifies the total number of Bell pairs required for the candidate solution's block configuration. This total encapsulates both gate teleportations within individual blocks and qubit teleportations across the network. The phenotype thus serves as a measure of the solution's effectiveness in optimising quantum communication. To evaluate the phenotype and thus the viability of each candidate solution, we introduce a fitness function, evaluateFitness: $\mathrm{P} \rightarrow \mathrm{N}$, that maps the individual to a natural number. In our case, this function is counting the total number of Bell pairs and hence teleportations are needed under the configuration under 5 consideration.

Through the iterative processes of selection, crossover, and mutation, our genetic algorithm seeks to evolve the population towards configurations that minimise Bell pair usage, thereby enhancing the efficiency and feasibility of DQC tasks. By continuously refining the genotypes within the population based on their fitness scores, the algorithm drives towards an optimal or near-optimal distribution of computational loads and quantum communication requirements across the network. This framework not only provides a method for optimising quantum network configurations but also offers insights into the trade-offs between computational depth and quantum communication resources, laying the groundwork for further innovations in DQC architectures.

3.3 *Optimised Distributed Quantum Circuit Execution via Meta-Heuristic Approach (ODQC-MHA)—A Detailed Description*

The Optimised Distributed Quantum Circuit Execution via Meta-Heuristic Approach (ODQC-MHA) makes use of a genetic algorithm formalism to find a minimum number of total Bell pairs required for a distributed execution of a given quantum circuit by finding an arrangement of block lengths that minimises the total cost. This section provides a detailed description of the ODQC-MHA components that were abstracted away in Fig. 3, complementing also the overview provided in Fig. 4.

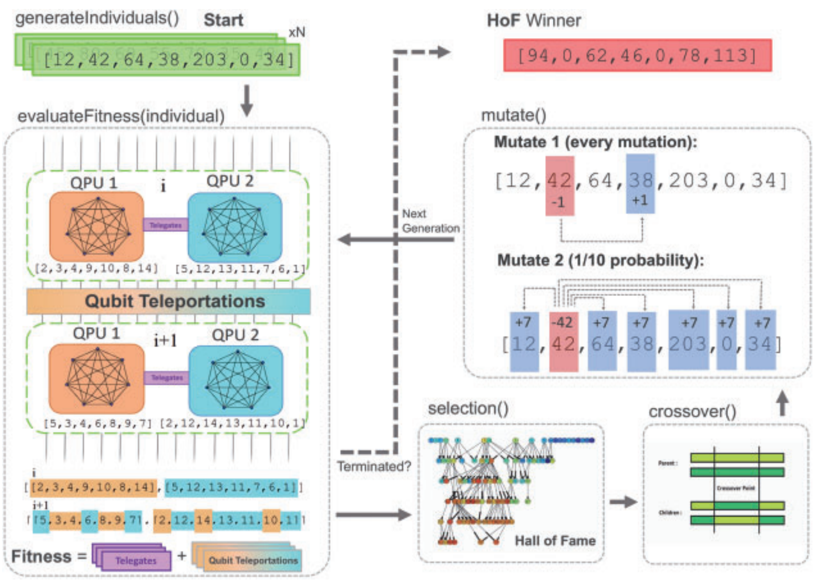


Figure 4: Structure of genetic algorithm. GenerateIndividuals(): create N lists, of a given size, representing the number of layers per block of a circuit. selection():'Hall of Fame' selection process ensures that the best individual to ever exist is chosen as the optimal. Crossover(): randomly chooses two individuals from the mating pool to create each new generation of superior individuals. Mutate: individuals are randomly chosen to mutate, mutate 1 occurs every mutation and mutate 2 occurs with probability 1/10 for each mutation. HoF Winner: the best individual that has existed throughout the generations is selected as the optimal 'blocking' solution

3.3.1 Efficient Qubit Teleportation Decisions

For each individual in the genetic algorithm, the quantum circuit is segmented into blocks based on the number of layers specified by the individual's genotype. To optimise the allocation of qubits across multiple QPUs, a method such as graph partitioning (e.g., GPA) is employed. This step aims to find an approximately optimal distribution of qubits over the available processors for each block. Given the allocation for each block, the challenge arises in transitioning qubits from their configuration in one block to the next. This transition is not straightforward due to the flexibility in QPU assignments: any given allocation might correspond to any QPU. This leads to a complex problem, especially as the number of QPUs increases, where finding the most efficient mapping between allocations in consecutive blocks becomes computationally intensive. To address this, we construct a bipartite graph $G = (I, J, L)$, where nodes in disjoint set I represent the processor allocations in block i, and nodes in set J represent the allocations in block $i + 1$ (Fig. 5). The edges in this graph, L, denote the potential mappings between allocations, with weights reflecting the minimum number of qubit differences (and thus qubit teleportations) required for each mapping. By negating these weights and applying a maximum weighted matching algorithm [29], we identify the mapping that minimises the number of qubit teleportations needed for reallocation between blocks. This approach significantly reduces the complexity of finding optimal qubit transitions between blocks.

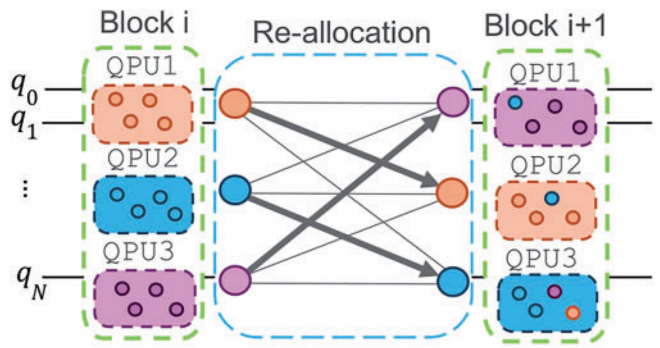


Figure 5: Bipartite graph representing the re allocation between multiple QPUs, it is clear to see that there are multiple ways to map a given allocation to QPU

3.3.2 Components of the Genetic Algorithm

Initialisation (generateIndividuals)—firstly the initial population is generated. Toward that goal, we generate homogeneous lists of a given length, which corresponds to the maximum number of blocks available in the solution. Then applying the mutate function (described later) to each individual 100,000 times we generate diverse genotypes for the initial population. It is important that these individuals are highly varied due to the size and complexity of the solution space.

Evaluation of Fitness (evaluateFitness)—the fitness function evaluates the efficiency of a given qubit allocation and teleportation scheme. It does so by summing the total number of gate teleportations within each block (determined by the initial qubit allocations) and the qubit teleportations between blocks (as optimised by the bipartite graph matching). The objective of the genetic algorithm is to minimise this sum, thereby reducing the overall quantum communication and computation overhead in the DQC framework. The evaluation process effectively quantifies the "cost" of a particular configuration of qubit allocations and transitions, guiding the genetic algorithm toward solutions that optimise the use of quantum resources. By focusing on minimising the combined total of gate and qubit teleportations, the algorithm seeks configurations that offer the best balance between computational efficiency and the practical constraints of DQC. This structured approach allows for a clear understanding of how qubit teleportation and allocation decisions impact the overall efficiency of quantum computing operations, providing a solid basis for optimising DQC architectures.

Crossover Function—the crossover function is the process of generating offspring from the selected parent genes. These offspring are generated such that they share some elements from either parent. The crossover function used is a simple two-point crossover which chooses a subset of random size from each pair of parents and swaps them. Since the genotype's values must sum up to the total number of layers in the quantum circuit, we employ a rebalancing routine to enforce that constraint after the crossover.

Mutation Function—The mutation function involves randomly choosing 2 indices i and j, $i \neq j$ in the individual, with a predefined probability of occurence. Defining a mutation constant, c, a fair coin is flipped and the mutation constant is either added to g_i and subtracted from g_j or *vice versa*. Also, each mutation has a probability p of introducing a zero element to the individual, by subtracting a randomly chosen indices value from itself and spreading the value among the remaining indices.

Selection Process—The selection process entirely replaces the parental population, requiring that the selection procedure is stochastic and allows the same individual to be selected more than once. We used the selTournament() function as part of the DEAP framework, which was found to be a suitable mutation function for searching the solution space thoroughly.

3.3.3 Overview

The genetic algorithm adjusts the size of each block—between which qubit teleportations are done to get from one allocation to the next—to try and minimise the total number of Bell pairs required. The size of a block is allowed to go to zero and if so, it is ignored by the calculation of teleportations. In other words, the optimal 'blocking' of the circuit may contain fewer blocks than the initial candidate solution. Note that this meta-heuristic can use any algorithm for the allocation of qubits inside a given block to calculate the Bell pairs needed for the gate teleportations. For example, if there are just two QPUs one could implement K-L algorithm and in a multi-QPU framework the proposed GPA is more suitable. In the next section, we implement ODQC-MHA using the DEAP python framework [30] to execute the genetic algorithm with these definitions, to converge on a close to optimal blocking of the circuit that required the minimum number of gate and qubit teleportations combined for a given quantum circuit. Hereafter, the term ODQC-MHA(K-L) refers to the meta-heuristic that applies the K-L algorithm to each block. Conversely, ODQC-MHA(GPA) denotes the variant where the GPA is employed for graph partitioning in each block.

4 Performance Evaluation

4.1 Pre-Processing

In this section, we describe the steps taken to analyse benchmark circuits to evaluate the performance of ODQC-MHA. Each circuit analysed is represented in a QASM (QUantum ASeMbley) [31] file that contains all of the logical instruction information in order. The QASM file is parsed, and the circuit is then represented as a directed acyclic graph (DAG). The DAG shows the dependencies of each gate and allows us to determine which operations can occur simultaneously (in the same layer). Given that we have the DAG, we can divide the circuit, by layer, into the blocks given by an individual's genotype Gp, as explained previously. For each block, an interaction matrix is constructed using the QASM instructions, this matrix is then used to build a network [32] graph object for which the standard graph partitioning algorithms can be used. ODQC-MHA can be applied to any circuit of any size, although the execution time scales with the number of qubits and the maximum number of blocks. Note that the bottleneck here is the number of qubits in the circuits as this determines the size of the graph to be partitioned.

For each comparison, the mean value for the number of combined gate and qubit teleportations when using *ODQC-MHA* for multiple trials on each circuit is taken to be the average performance of the algorithm. The ratio of this mean value to the number of teleportations when using K-L (for two QPUs) and GPA (for > 2 QPUs), with no circuit blocking, is presented as the percentage improvement of ODQC-MHA. The maximum number of QPUs considered in this study is three, this is because the main comparison presented is with the commonly used K-L algorithm, which applies only to the two QPU cases. ODQC-MHA is agnostic to any efficient graph partitioning algorithm for > 2 QPUs, within each block.

4.2 Non-Random Quantum Circuits (2 QPUs)

Initially, we ran ODQC-MHA(K-L) on quantum circuits from QASMBench [33]. Information about the benchmark circuits used is shown in Table 1. The results of this analysis are shown in Fig. 6a, showing the percentage improvement over using K-L for the entire circuit, i.e., no qubit teleportations. We compare three configurations of ODQC-MHA(K-L), with the maximum allowed numbers of blocks (MAB) of; 10, 50, and 100.

Table 1: Benchmark quantum circuits for Section IV-B

ID	Circuit name	Qubits	Depth (CX only)	Unary gates	CX gates
1	Adder_n118	118	4	1107	845
2	sym9_146	12	91	180	148
3	cycle10_2 110	12	3386	3402	2648
4	Inc_237	16	3463	5983	4636
5	cm85a_209	14	3818	6428	4986
6	rd84_253	12	4466	7698	5960
7	root_255	13	5354	9666	7493
8	mlp4_245	16	6190	10,620	8232
9	clip_206	14	10,734	19,055	14,772
10	Dist_223	13	11,911	21,422	16,624

While the improvement varies across the circuits, we see a clear trend. For increasing depth there is a region for which 100 MAB shows the smallest improvement while 10 MAB shows the largest, in the central region we see that 50 MAB shows the most improvement, and for the higher end of circuit depth, 100 MAB. We identify a likely reason for this trend, for smaller circuits it is possible to 'overblock', that is, it becomes hard to converge on a solution—on average—due to the starting size of an individual genotype. It is worth noting that we believe this analysis on small benchmark circuits is arbitrary because the performance of the heuristic depends strongly on the distribution of CNOT gates, in the next section, we discuss the performance of the algorithm on average, using large randomly generated circuits.

The inconsistent performance between different configurations across different circuits suggests a limitation with the design of ODQC-MHA. Future work might include optimising the allowed size of an individual, the population size, and the number of generations, depending on the circuit at hand. This should allow the algorithm to search the given solution space more efficiently and prevent converging on local minima.

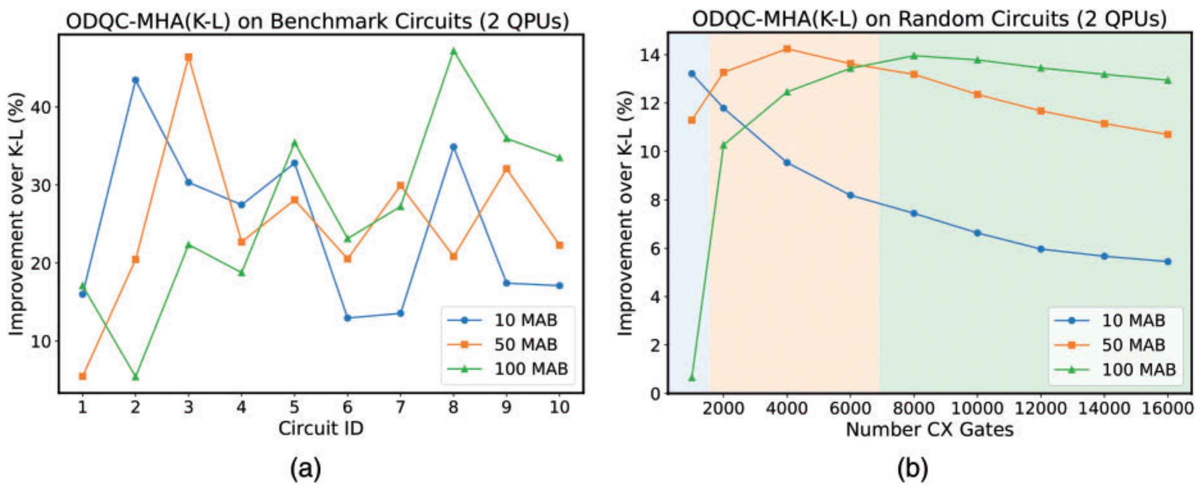


Figure 6: (a) Comparison percentage improvement for ODQC-MHA(KL) on benchmark circuits for varying maximum allowed number of blocks (MAB) (10, 50, 100), across 2 QPUs. The benchmark circuits are ordered by increasing circuit depth. Each data point is the mean of 100 executions of ODQC-MHA(K-L). Circuit ID I; (b) Comparison of ODQC-MHA(K-L) for varying maximum allowed number of blocks (10, 50, 100), to ODQC-MHA(K-L) for 1 block on randomly generated circuits, distributed over 2 processors. Each data point is the mean value of multiple runs of ODQCMHA(K-L) on different randomly generated, 16 qubit, circuits. The percentage difference is plotted

4.3 Random Quantum Circuits (2 QPUs)

To demonstrate the effectiveness of ODQC-MHA on average, we ran qubit allocation on randomly generated circuits containing only CNOT gates, which are the only universal gates that are considered in the algorithm. This demonstrates the average behaviour on large circuits where the distribution of CNOT gates tends towards homogeneity. In principle, any bias in the distribution of CNOTs should be exploited by a good heuristic method.

Firstly, we generated random, 16 qubit circuits of varying numbers of CNOT gates, in the same range as the benchmark circuits used. We ran ODQC-MHA(K-L) to allocate across 2 QPUs for each circuit. This analysis was done 100 times, and the average performance was plotted in Fig. 6b. This was done for different initial configurations of ODQC-MHA(K-L), allowing for a maximum number of blocks (length of genotype) of 10, 50 and 100. The genetic algorithm could—on average—converge on a solution with fewer bell pairs required across all the randomly generated circuits, with a general trend towards smaller improvement for an increasing number of gates. However, we again observed the same trend as explained before, a region where each configuration performed best. This effect is shown dramatically in the first point (left). This indicates that ODQC-MHA(K-L) requires further optimisation to select an optimal number of blocks. Due to resource limitations, we are not able to analyse the limit that an improvement is shown by increasing the allowed maximum number of blocks.

Next, we generated random circuits of 8, 16, and 32 qubits with up to 100,000 CX gates, in order to test the performance of ODQC-MHA(K-L) on larger circuits. The results are plotted in Fig. 7a. We saw a similar improvement trend across different numbers of gates. Interestingly, the reduction in the number of bell pairs increases for a greater number of qubits. This could be because, for the same number of CNOT gates across more qubits, the interactions are more sparsely distributed and will likely have shorter depth.

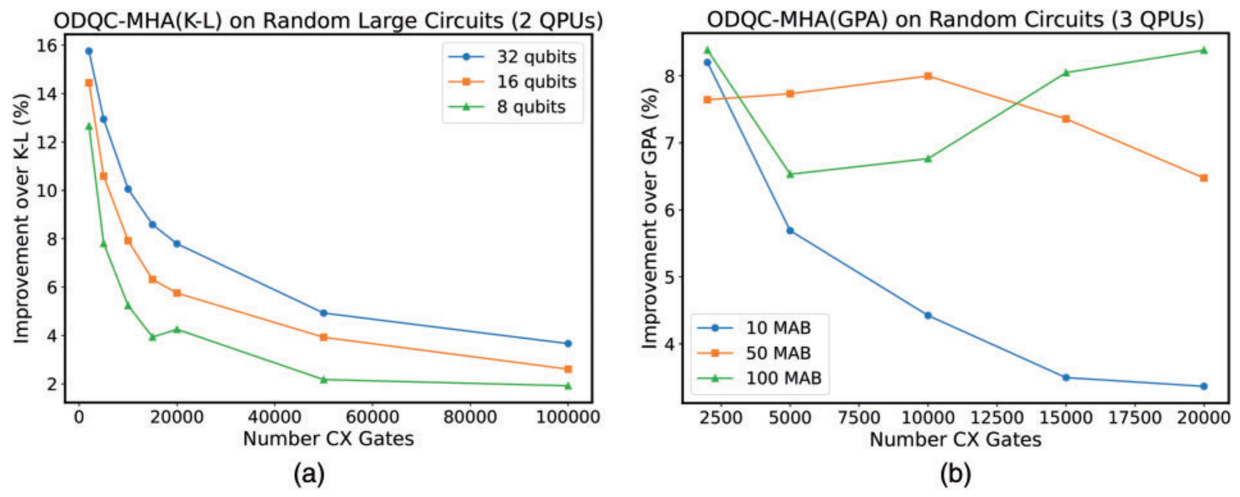


Figure 7: (a) Comparison of ODQC-MHA(K-L) to K-L on randomly generated circuits of varying number of qubits (8, 16, 32) on randomly generated circuits, across 2 QPUs; (b) Comparison of ODQC-MHA(GPA) to GPA on randomly generated 16 qubit circuits, for different maximum allowed number of blocks (10, 50, 100)

4.4 Random Quantum Circuits (3 QPUs)

Finally, to demonstrate the performance of ODQC-MHA using a different heuristic for graph partitioning within each block, we performed the same analysis but distributed each circuit across 3 QPUs using ODQC-MHA(GPA) within each block, presented in Fig. 7b. Here we observe a similar trend as for 2 QPUs, however, the first value for maximum allowed blocks of 100 shows anomalously high improvement. We can attribute this to the fact that the smallest circuits have larger variance in the distribution of CX gates and so the genetic algorithm may get ‘lucky’ on certain circuit configurations. In the future, this analysis could be extended to greater than 3 QPUs.

4.5 Discussion

In the limit that the number of gates is large, we would expect that the performance of the K-L algorithm at allocating the qubits across two processors would tend to the performance of randomly allocating the qubits across two processors. This would mean that half of the CNOT gates are executed remotely. However, for any finite circuit depth, the optimal solution will always be better than half of the CNOT gates. This also applies to our metaheuristic, because the genetic algorithm can always find a block length where the performance of K-L is better than half and so—if allowed to execute properly—will be able to exploit sections of the circuit where K-L performs well at allocating the qubits.

5 Conclusion

Compiling circuits for DQC will be paramount to the future and scaling of quantum computers, within a ‘quantum data center’. In this letter we addressed the problem of qubit allocation within a compilation, using a meta-heuristic which optimises for the minimum number of qubit and gate teleportations. The results when comparing our method to the standard method of graph partitioning using the K-L algorithm show a significant improvement.

Acknowledgement: The authors would like to thank Luca Della Chiesa at Cisco for his valuable insights and feedback during this work.

Funding Statement: The authors received no specific funding for this study.

Author Contributions: The authors confirm contribution to the paper as follows: study conception and design: Oliver Crampton, Panagiotis Promponas, Paul Polakos, Louis Samuel; data collection: Oliver Crampton, Panagiotis Promponas, Richard Chen, Paul Polakos, Louis Samuel; analysis and interpretation of results: Oliver Crampton, Panagiotis Promponas, Richard Chen, Paul Polakos, Louis Samuel; draft manuscript preparation: Oliver Crampton, Panagiotis Promponas, Richard Chen, Leandros Tassiulas. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: Data not available due to commercial restrictions.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest to report regarding the present study.

References

1. Grover LK. A fast quantum mechanical algorithm for database search. In: Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing; 1996 May 22–24; Philadelphia, PA, USA. p. 212–9. doi:10.1145/237814.237866.
2. Shor PW. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Rev. 1999;41(2):303–32. doi:10.1137/S0036144598347011.
3. Skosana U, Tame M. Demonstration of shor's factoring algorithm for n = 21 on ibm quantum processors. Sci Rep. 2021;11(1):16599. doi:10.1038/s41598-021-95973-w.
4. Brown KL, Munro WJ, Kendon VM. Using quantum computers for quantum simulation. Entropy. 2010;12(11):2268–307. doi:10.3390/e12112268.
5. Daley AJ, Bloch I, Kokail C, Flannigan S, Pearson N, Troyer M, et al. Practical quantum advantage in quantum simulation. Nature. 2022;607(7920):667–76. doi:10.1038/s41586-022-04940-6.
6. Van Meter R, Devitt SJ. The path to scalable distributed quantum computing. Computer. 2016;49(9):31–42. doi:10.1109/MC.2016.291.
7. Nielsen MA, Chuang IL. Quantum computation and quantum information. Cambridge, UK: Cambridge University Press; 2001. doi:10.1017/CBO9780511976667.
8. Deutsch D. Quantum theory, the Church–Turing principle and the universal quantum computer. Proc R Soc Lond. A Math Phys Sci. 1985;400(1818):97–117. doi:10.1098/rspa.1985.0070.
9. Barenco A, Bennett CH, Cleve R, DiVincenzo DP, Margolus N, Shor P, et al. Elementary gates for quantum computation. Phys Rev A. 1995;52(5):3457. doi:10.1103/PhysRevA.52.3457.
10. Einstein A, Podolsky B, Rosen N. Can quantum-mechanical description of physical reality be considered complete? Phys Rev. 1935;47(10):777. doi:10.1103/PhysRev.47.777.
11. Bennett CH, Brassard G, Crepeau C, Jozsa R, Peres A, Wootters WK. Teleporting an unknown quantum state via dual classical and einstein-podolsky-rosen channels. Phys Rev Lett. 1993;70(13):1895. doi:10.1103/PhysRevLett.70.1895.
12. Botea A, Kishimoto A, Marinescu R. On the complexity of quantum circuit compilation. In: Proceedings of the International Symposium on Combinatorial Search; 2018 Jul 14–15; Stockholm, Sweden. Vol. 9, p. 138–42. doi:10.1609/socs.v9i1.18463.
13. Moro L, Paris MG, Restelli M, Prati E. Quantum compiling by deep reinforcement learning. Commun Phys. 2021;4(1):178. doi:10.1038/s42005-021-00684-3.
14. Zhu P, Cheng X, Guan Z. An exact qubit allocation approach for NISQ architectures. Quantum Inf Process. 2020;19(11):391. doi:10.1007/s11128-020-02901-4.

15. Houshmand M, Mohammadi Z, Zomorodi-Moghadam M, Houshmand M. An evolutionary approach to optimizing teleportation cost in distributed quantum computation. Int J Theor Phys. 2020;59(4):1315–29. doi:10.1007/s10773-020-04409-0.
16. Daei O, Navi K, Zomorodi-Moghadam M. Optimized quantum circuit partitioning. Int J Theor Phys. 2020;59(12):3804–20. doi:10.1007/s10773-020-04633-8.
17. Mao Y, Liu Y, Yang Y. Qubit allocation for distributed quantum computing. In: IEEE INFOCOM 2023-IEEE Conference on Computer Communications; 2023 May 17–20; New York City, NY, USA. doi:10.1109/INFOCOM53939.2023.10228915.
18. Finigan W, Cubeddu M, Lively T, Flick J, Narang P. Qubit allocation for noisy intermediate-scale quantum computers. arXiv:1810.08291. 2018. doi:10.48550/arXiv.1810.08291.
19. Ash-Saki A, Alam M, Ghosh S. Qure: Qubit re-allocation in noisy intermediate-scale quantum computers. In: Proceedings of the 56th Annual Design Automation Conference 2019; 2019 Jun 2–6; Las Vegas, NV, USA. p. 1–6. doi:10.1145/3316781.3317888.
20. Davis MG, Chung J, Englund D, Kettimuthu R. Towards distributed quantum computing by qubit and gate graph partitioning techniques. arXiv:2310.03942. 2023.
21. Bandic M, Prielinger L, Nußlein J, Ovide A, Rodrigo S, Abadal S, et al. Mapping quantum circuits to modular architectures with qubo. arXiv:2305.06687. 2023. doi:10.1109/QCE57702.2023.00094.
22. Nikahd E, Mohammadzadeh N, Sedighi M, Zamani MS. Automated window-based partitioning of quantum circuits. Phys Scr. 2021;96(3):035102. doi:10.1088/1402-4896/abd57c.
23. Kernighan BW, Lin S. An efficient heuristic procedure for partitioning graphs. Bell Syst Tech J. 1970;49(2):291–307. doi:10.1002/j.1538-7305.1970.tb01770.x.
24. Fiduccia CM, Mattheyses RM. A linear-time heuristic for improving network partitions. In: Papers on twenty-five years of electronic design automation. New York, NY, USA: Association for Computing Machinery; 1988. p. 241–7. doi:10.1145/62882.62910.
25. Karypis G, Kumar V. Multilevel k-way hypergraph partitioning. In: Proceedings of the 36th Annual ACM/IEEE Design Automation Conference; 1999 Jun 21–25; New Orleans, LA, USA. p. 343–8. doi:10.1145/309847.309954.
26. Childs AM, Schoute E, Unsal CM. Circuit transformations for quantum architectures. arXiv:1902.09102. 2019. doi:10.48550/arXiv.1902.09102.
27. Abelem A, Towsley D, Vardoyan G. Quantum internet: the future of internetworking. arXiv:2305.00598. 2023. doi:10.48550/arXiv.2305.00598.
28. Katoch S, Chauhan SS, Kumar V. A review on genetic algorithm: past, present, and future. Multimed Tools Appl. 2021;80(5):8091–126. doi:10.1007/s11042-020-10139-6.
29. Schrijver A. Combinatorial optimization: polyhedra and efficiency. Berlin/Heidelberg, Germany: Springer; 2003. Vol. 24.
30. De Rainville FM, Fortin FA, Gardner MA, Parizeau M, Gagne C. DEAP: a python framework for evolutionary algorithms. In: Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation; 2012 Jul 7–11; Philadelphia, PA, USA. p. 85–92. doi:10.1145/2330784.2330799.
31. Cross AW, Bishop LS, Smolin JA, Gambetta JM. Open quantum assembly language. arXiv:1707.03429. 2017. doi:10.48550/arXiv.1707.03429.
32. Hagberg A, Swart P, Chult DS. Exploring network structure, dynamics, and function using network. Los Alamos, NM, USA: Los Alamos National Lab. (LANL); 2008. doi:10.25080/TCWV9851.
33. PNNL. QASMBench: a low-level OpenQASM benchmark suite for NISQ evaluation and simulation. Please see our paper for details [Internet]. [cited 2025 Jan 1]. Available from: https://github.com/pnnl/QASMBench.