Distributing Quantum Circuits Using Pre-Distributed Entanglement Pairs over Quantum Networks

Yukun Yang, Ranjani G. Sundaram, Himanshu Gupta
Department of Computer Science, Stony Brook University, Stony Brook, NY 11790, USA

***Abstract*—Distributing quantum circuits across a network of quantum devices is a recent approach to improve the scalability of quantum computation. Entangled pairs of qubits distributed across network nodes are used as a communication resource for executing inter-node (remote) gates. However, establishing these entanglement pairs incurs significant latency, and doing so during the circuit's execution may lead to the decoherence of logical qubits. Instead, it may be beneficial to distribute certain entanglement pairs before the circuit is executed. Thus, we consider the problem of distributing a quantum circuit across a quantum network containing some pre-distributed entanglement pairs and seek to minimize the execution time of the circuit.**

We address this problem in three steps. First, given the quantum network and (optionally) a probability distribution over training circuits, we determine a set of pre-distributed entanglement pairs under a specified budget. Second, we compute a qubit-to-memory allocation that minimizes the expected execution time given these pre-distributed resources. Finally, we identify any additional entanglement pairs required during execution, generate them concurrently, and execute the circuit gates. We evaluate our methods on a broad range of benchmark circuits.

I. Introduction

The distribution of quantum circuits across a network of quantum devices is a recent approach to improving the scalability of quantum computation. By connecting smaller quantum processors through quantum and classical communication channels, it becomes possible to perform computations (via remote gates) that would exceed the capacity of any single device. However, maintaining a quantum channel requires replenishing entangled pairs (EPs) across adjacent network nodes. Establishing these entanglement pairs during the execution of a circuit can incur significant latency and lead to the decoherence of qubits. Thus, it is preferable to exploit existing EPs that have been distributed before the execution of the circuit. Therefore, in this work, we distribute a given quantum circuit across a quantum network and seek to minimize circuit execution time by exploiting pre-distributed EPs and generating required EPs with minimum latency.

In particular, before execution begins, we allocate a budget to actively generate and refresh a subset of EPs across the network. These pre-distributed EPs are chosen strategically, based on the network topology and (optionally) a probability distribution of anticipated circuits, so that frequently used or high-cost remote connections are established in advance. Unlike replenishment-based approaches, these EPs are one-time resources that are consumed during circuit execution, making their placement and distribution a critical design choice.

The distribution of quantum circuits then involves three stages: (i) generate pre-distributed EPs across the network under the given budget, (ii) map the qubits of the circuit to the qubit memories of the quantum network to minimize the expected execution time given the pre-distributed resources, and (iii) execute the circuit by generating any additional EPs concurrently and consuming these newly created pairs and existing ones to perform remote gate operations. Our proposed approaches are designed to maximize the utilization of pre-distributed EPs while minimizing the latency of additional entanglement generation during execution.

Prior Work. The problem of distributing quantum circuits in quantum networks has gained significant attention in recent years, resulting in the development of efficient solutions tailored to various settings and objectives. Most prior work on distributing quantum circuits has focused on the objective of minimizing the number of maximally-entangled pairs (EPs) either by minimizing the number of cat-entanglements [1, 2] or the number of teleportations [3–5]. Several recent works [6–10] have focused on the more significant objective of minimizing the total latency incurred in generating the EPs needed for executing the remote gates. However, all of them only start generating EPs *during* circuit execution and do not consider cases where pre-distributed EPs are available for consumption in the network. In addition, Fan et al. [11, 12] studied the efficient distribution and purification of GHZ and graph states in quantum networks. Although we also evaluate GHZ generation as a benchmark circuit, these works focus on state distribution, whereas we address distributed circuit execution with pre-distributed EPs.

Our Work. In this paper, we formulate the problem of distributing a quantum circuit across a network with pre-distributed entanglement pairs (EPs) to minimize its execution time under network and decoherence constraints. Our contributions are threefold:

(i) *Pre-distributed EPs allocation.* We design and analyze multiple algorithms for selecting where to place a limited pre-distribution budget, guided by training circuits and network topology.

(ii) *Qubit-to-memory allocation.* We extend and adapt classical optimization techniques (including greedy heuristics, an extended quadratic assignment formulation, and simulated annealing) to assign circuit qubits to network memories given the pre-distributed EPs.

(iii) *Execution with additional EPs.* We integrate these

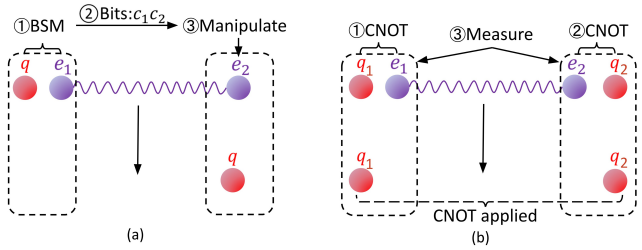

Fig. 1. Quantum communication. (a) Teleportation and (b) Telegate. Dashed boxes are the network nodes; the initial (final) state of the qubits is at the top (bottom).

allocations into a full execution model that accounts for decoherence and on-demand generation, ensuring correctness while minimizing latency.

II. Background and Models

Quantum Circuit Representation. We represent an *abstract quantum circuit* C over a set of qubits $\mathcal{Q} = \{q_1, q_2, \ldots\}$ as a sequence of gates $\langle g_1, g_2, \ldots\rangle$ where each g_t is a binary CNOT gate or a unary gate (a universal set of gates). We represent binary gates as triplets (q_i, q_j, t) where q_i and q_j are the two operands and t is the time instant of the gate in the circuit, and unary gates as pairs (q_i, t) where q_i is the operand and t is the time instant.

Quantum Network (QN). A quantum network is a network of quantum computers (QCs) represented as a connected graph with nodes as QCs and edges representing (quantum and classical) communication links. Each computer has a certain amount of quantum memory to store the data/circuit qubits.

Distribution of Quantum Circuits over QN. Given a quantum network (QN) and a quantum circuit, we seek to distribute and execute the given circuit over the network in such a way that the execution incurs minimum time. Distribution of a circuit over a network essentially entails two aspects: (i) distributing the circuit qubits across the QCs/nodes of the QN, and (ii) executing the "remote" binary gates (i.e., gates with operands on different QCs) efficiently. The main overhead in distributed execution comes from these remote gates, which require EPs to be available across the involved nodes.

Executing Remote Quantum Gates. To execute such remote gates, we need to bring all operands' values into a single QC via quantum communication. Since direct transmission of qubits is subject to unrecoverable errors, we consider the following ways to communicate qubits across network nodes.

Teleportation. An alternative approach to physically transmitting a qubit from a node A to B is by *teleportation* [13] which requires a priori distribution of an EP over A and B. See Fig. 1(a). Teleportation moves the qubit state so that the binary gate can be executed locally at the destination QC.

Telegates. Another approach is the telegate protocol [6], which executes a remote CNOT gate between operands at A and B using only local operations and classical communication. Telegates also require a priori EP between A and B. See Fig. 1(b).

Teleportation vs. Telegates. Both communication protocols require a single EP. Teleportation transfers the qubit state to another node, while the telegate protocol executes the remote gate directly without qubit movement. In this work, we focus on telegates for executing remote gates, as they align naturally with a static qubit-allocation function. The use of teleportations allows dynamic qubit allocations, but requires a more complex formulation, which we defer to future work.

III. Problem Formulation

We start by defining some terms and models before moving on to the problem formulation.

Qubit-Allocation Function. Consider a quantum network with nodes (quantum computers) denoted as $\mathbb{P} = \{P_1, P_2, \ldots, P_N\}$, where each node P_i contains a set of qubit memories $M(P_i)$. For a quantum circuit C, let $Q(C)$ represent its set of qubits. To execute C on the network, we define a *qubit allocation function* $\eta : Q(C) \to \bigcup_i M(P_i)$, which assigns each circuit qubit to a distinct qubit memory at the network node. Specifically, η must be one-to-one, ensuring that each qubit memory is assigned to at most one circuit qubit.

Remote and Local Gates. Given η, a gate (q_i, q_j, t) in C is classified as *remote* if its operands q_i and q_j are assigned to different nodes, i.e., $\eta(q_i) \in M(P_u)$ and $\eta(q_j) \in M(P_v)$ with $u \neq v$. In contrast, a gate is *local* if both operands reside at the same node.

Pre-distributed EPs in Network. An important aspect in executing remote gates is how the entanglement resources are provisioned within the network. In our model, the network is initialized with a limited budget of *pre-distributed EPs* before execution begins. These EPs are consumable one-time resources that can be consumed immediately during execution, thus reducing the need for online entanglement generation and minimizing overall execution latency. Since EPs are subject to decoherence, the network protocol refreshes the pre-distributed EP pool during idle periods (when the network is not executing a circuit), so that the provisioned resources remain valid and available at the start of execution. During circuit execution, this background refresh is paused; only on-demand entanglement generation needed to cover unmet demand is performed. We assume that consuming the pre-distributed EPs (i.e., retrieving and using an already-established entangled pair during execution) incurs negligible online latency compared to on-demand entanglement generation and entanglement swapping.

Coupling Graph and Swap Operations. Each node P_i has a *coupling graph* U_i that models the connectivity among its qubit memories. An edge (m_a, m_b) in U_i indicates adjacent qubits capable of direct gate execution. For non-adjacent qubits m_c and m_d in P_i, executing a gate requires swap operations along the shortest path between them in U_i. The number of swaps equals the path length in U_i.

Static Qubit-Allocation Enforcement. In general, the qubit-allocation function can change during circuit execution due to teleportations and/or swap operations. However, to simplify

algorithm design and analysis, we enforce a static η throughout execution. This implies:

1) *Remote gates.* We execute remote gates exclusively via telegates, and do not teleport data qubits between nodes. Allowing teleportation-based routing would make η time-varying and requires modeling dynamic mappings, which is outside our scope.
2) *Local gates with swaps.* When a local gate requires swaps, we swap qubits to bring them together, apply the gate, and then reverse the swaps to restore the original allocation.

Distributed Quantum Computation with Pre-distributed EPs (DQC−PD) Problem. Given a quantum circuit C, a quantum network, and a pre-distribution budget K, the DQC−PD problem is to execute C across the network with minimum total execution time, subject to network constraints such as limited quantum memories at each node, entanglement generation resources, and decoherence. The DQC−PD problem has three key components: (i) determining how to allocate the limited pre-distribution budget across the network, (ii) computing a static qubit-allocation function η that jointly accounts for the pre-distributed EPs and on-demand entanglement requirements, and (iii) devising an execution scheme for local and remote gates given all the above, so that the overall circuit execution time is minimized.

Remarks and Computational Complexity. Our formulation differs fundamentally from previous work due to the presence of pre-distributed EPs. Instead of beginning directly with qubit allocation, we must first decide how to allocate a limited pre-distribution budget and then compute the qubit allocation function accordingly. This joint optimization of entanglement allocation and qubit placement leads to a combinatorial problem that is computationally intractable. Even when the set of pre-distributed EPs is fixed, determining the optimal qubit-to-memory assignment requires jointly reasoning about all pairwise gate interactions and network distances, a structure analogous to the classical quadratic assignment problem, which is NP-hard. Consequently, exact optimization is infeasible for realistic circuit and network sizes, motivating the development of efficient heuristic algorithms that approximate optimal allocation and execution decisions in the subsequent sections.

IV. Related Works

Minimizing the Number of EPs. The DQC problem is to distribute a given quantum circuit over a quantum network with some optimization objective. In contrast to the DQC−PD problem, the DQC problem ignores the coupling graphs within each computer [1, 2, 4, 5, 14, 15], and thus the needed swap operations. The optimization objective used in almost all of these works has been to minimize the communication cost, defined as the *number* of EPs used; a couple of recent works [16, 17] consider EPs with arbitrary (but fixed) cost and develop a simulated-annealing heuristic for the qubit-allocation part of the DQC problem.

Minimizing Circuit Execution Time. The works close to ours [6–10] address the DQC problem with minimizing circuit execution time, but all assume that the EPs required are generated during circuit execution and do not consider pre-distributed EPs. [7] focuses on estimating only the *worst-case* overhead in executing a circuit over a network, by the worst-case linear network topology; the overhead considered is in terms of an increase in circuit "layers" and EPs required to execute remote gates. [6] focuses on the efficient execution of remote gates using telegates, *given* a qubit allocation (they consider qubit allocation to be out of the scope of their work), to minimize the number of circuit layers; they assume generation latency of each EP to be uniform (a constant), and ignore decoherence constraints. [8] considers the more general *multi-circuit* setting, where several circuits are executed concurrently over the same network; they design scheduling and resource allocation schemes to minimize the overall execution time across circuits. [9, 10] are the works closest to ours, tackling the DQC problem in three steps-allocating qubits to memories, selecting teleportations, telegates, and cat-entanglements for remote gate execution, and generating the necessary EPs. The DQC−PD problem is an extended formulation of their DQC problem and considers pre-distributed EPs.

Pre-Distribution Framework. Our problem formulation also differs fundamentally from the pre-distribution framework of Ghaderibaneh et al. [18]. Their model selects super-links in the network to continuously replenish entanglement, thereby reducing the latency of future EP requests across arbitrary node pairs. In contrast, our goal is to execute a specific quantum circuit with the help of pre-distributed EPs, which are provisioned before each execution and treated as one-time consumable resources. This setting does not model a continuous replenishment process during circuit execution; instead, resources are provisioned (and refreshed) between executions and then consumed during execution. The central challenge then becomes how to allocate a limited pre-distribution budget so that the subsequent qubit allocation and circuit execution incur minimal latency.

V. High-Level Approach

In the previous section, we formulated the DQC−PD Problem as minimizing circuit execution time over a network given a limited pre-distribution budget. We use a three-step approach, consisting of allocating pre-distributed EPs, determining the qubit-allocation function, and finally determining the execution scheme.

1) *Allocate Pre-distributed EPs.* Given a quantum network, a budget K of pre-distributed EPs, and (optionally) a distribution of training circuits, we determine how to allocate the EPs across memory pairs in the network to maximize their impact in reducing the overall execution time.
2) *Determine the Qubit-Allocation Function.* Given a quantum circuit, the network, and the chosen set of pre-distributed EPs, we compute the qubit-allocation function η that maps circuit qubits to network memories in a

way that minimizes the expected circuit execution time. Together with Step 1, this forms the primary focus of our work.

3) *Determine the Execution Scheme.* With the qubit-allocation function η fixed and the set of pre-distributed EPs available, we devise an execution scheme that involves determining when and how to generate the additional EPs required to execute the remote gates. The goal is to minimize the total execution time.

The final step largely follows prior work [9]: given a qubit allocation, we identify the remote gates, distinguish those already covered by pre-distributed EPs, and determine the set $\mathcal{E}$ of additional EPs that must be generated during execution.

Handling Decoherence. Note that in the presence of decoherence, EPs can decay both during and after their generation while waiting to be consumed. To mitigate this, we may need to partition $\mathcal{E}$ into subsets of EPs (called *batches*) with each batch generated independently, as generating the entire set concurrently could cause some EPs to exceed the decoherence threshold τ. We use the greedy algorithm from [9] to construct a sequence of batches, each of which can be generated concurrently without violating the decoherence constraints.

The pre-distributed EPs may also decohere during execution. To mitigate decoherence, the network refreshes the pre-distributed EP pool between executions / during idle periods, so that provisioned EPs remain valid at the start of each execution. Together with the batching strategy for additional EPs, this jointly ensures that decoherence does not compromise the accuracy of execution.

VI. Pre-distributed EPs Allocation

We now address the problem of determining how to allocate a limited pre-distribution budget across the network. This allocation step determines the matrix C over memory pairs, which directly influences the subsequent qubit-allocation and execution steps. Recall that the pre-distributed EPs are prepared before execution and are treated as instantly consumable.

Pre-distributed EPs Allocation Problem Formulation. Given a quantum network, a budget K of pre-distributed EPs, and (optionally) a probability distribution over circuits, we construct the following graphs and matrices.

Circuit Graph G_C. The circuit graph is an edge-weighted graph defined over the qubits $Q(C)$ of a given circuit C, with weight $w(q_i, q_j)$ on the edge (q_i, q_j) equal to the number of binary gates between q_i and q_j.

Network-Coupling Graph G_N. The network-coupling graph is an edge-weighted graph defined over the set of qubit memory $\bigcup_i M(P_i)$ in the given quantum network with quantum computers $\{P_i\}$. For two memories u, v, the weight $w'(u, v)$ is defined as follows:

- If $u, v \in M(P_i)$ for some computer P_i, then $w'(u, v)$ equals the swap latency times twice the shortest path distance between u and v on the coupling graph of P_i. Note that this represents the time it takes to execute the gate over qubits in u and v using swap operations.
- If $u \in M(P_i)$ and $v \in M(P_j)$ for $i \neq j$, then $w'(u, v)$ is the latency of independently generating an EP between the network nodes P_i and P_j.

EP Allocation Graph G_E. The EP allocation graph is defined over the same set of memories, with weight $w''(u, v)$ equal to the number of pre-distributed EPs between memories u and v. The budget constraint is

$$\sum_{u<v} w''(u, v)\, w'(u, v) \leq K.$$

Qubit-Allocation Function η. As introduced earlier in Sec. III, a qubit-allocation function $\eta : Q(C) \to \bigcup_i M(P_i)$ assigns each circuit qubit to a unique memory in the network.

Pre-distributed EPs Allocation Problem Formulation. Equivalently, let A denote the adjacency matrix of the circuit graph G_C, B denote the adjacency matrix of the network-coupling graph G_N, and C denote the adjacency matrix of the EP allocation graph G_E. For a given η, the execution cost of circuit A is

$$J(A, C, \eta) \;=\; 2 \sum_{i<j} \max\big(A_{ij} - C_{\eta(i),\eta(j)}, 0\big)\, B_{\eta(i),\eta(j)}.$$

Thus, interactions covered by C do not incur additional online latency, and only the unmet demand $\max\big(A_{ij} - C_{\eta(i),\eta(j)}, 0\big)$ contributes to the execution cost. For a distribution of representative circuits $\{A_r\}$ with probabilities $\{p_r\}$, the expected execution cost under allocation C is

$$\texttt{COST}(C) \;=\; \sum_r p_r \min_\eta J(A_r, C, \eta).$$

The pre-distributed EPs allocation problem is to select a feasible allocation matrix C that minimizes $\texttt{COST}(C)$ subject to the budget constraint above. Intuitively, a "good" allocation places EPs on memory pairs that are either structurally central in the network or highly demanded by the expected workload, thereby reducing the need for costly on-demand entanglement generation.

Assumption: Negligible Latency to *Consume* a Pre-distributed EP. Our execution-latency objective accounts for *on-demand* entanglement generation as the dominant time component, and treats the pre-distributed EP as an already-established resource that can be consumed immediately when executing a remote operation. Concretely, for a memory pair (u, v), if the pre-distributed supply $C_{u,v}$ covers the circuit demand, then those covered remote interactions incur no additional online entanglement-generation latency. Only the deficit must be generated during execution, which is captured by the term $\max\big(A_{ij} - C_{\eta(i),\eta(j)}, 0\big)$ in $J(A, C, \eta)$. This matches our system model in which the network is initialized with a limited pre-distribution budget (before execution begins), and these EPs are consumable one-time resources that reduce the need for online generation.

In the following, we present three algorithms for solving the pre-distributed EPs allocation problem: `Network-C`,

`Greedy-C`, and `SGD-C`. They differ in their information requirements: `Network-C` uses only the network topology, while `Greedy-C` and `SGD-C` exploit additional knowledge of representative training circuits (or anticipated workload distributions).

Algorithm 1. Network-based EP Allocation (Network-C). The `Network-C` algorithm is an efficient topology-based heuristic that allocates pre-distributed EPs using only the structural properties of the network. It does not require workload or circuit information. The intuition is to favor memory pairs that are well-connected (high degree), close in distance, and uncongested. The steps are as follows:

1) *Initialization.* Start with $C = 0$ and set all link loads $L_e = 0$ for edges e on the network. Here L_e denotes the number of pre-distributed EPs whose allocation paths traverse the edge e.
2) *Score Computation.* For each memory pair (u, v), compute the score
$$s_{u,v} = \frac{\deg(u) + \deg(v)}{B_{u,v}\,(1 + C_{u,v})\,(1 + L_{u,v})},$$
where $\deg(\cdot)$ is the degree in G_N, $B_{u,v}$ is the distance of the network, $C_{u,v}$ is the current number of EPs allocated on (u, v), and $L_{u,v}$ is the *congestion of the path*, defined as the sum of L_e over all edges e on the shortest path between u and v.
3) *Selection.* Identify the pair (u, v) with the highest score $s_{u,v}$, assign one EP to it, update $C_{u,v}$, and increase L_e along the shortest path between u and v.
4) *Iteration.* Repeat Steps 2–3 until the budget is exhausted.

By penalizing pairs whose shortest path is already heavily used, the congestion term $L_{u,v}$ ensures that allocations are spread more evenly across the network and avoids overloading specific links.

Algorithm 2. Greedy EP Allocation (Greedy-C). The `Greedy-C` algorithm assigns EPs to some top-ranked memory pairs in each round. Unlike `Network-C`, it explicitly leverages information from representative circuits (or anticipated workloads) to prioritize pairs with the highest unsatisfied demand. The steps are as follows:

1) *Initialization.* Start with $C = 0$ and compute the optimal qubit allocation function η_r for each training circuit A_r[1].
2) *Score remote pairs.* For each remote pair (u, v), compute its *positive remainder mass*
$$ss_{u,v} = \sum_r p_r \cdot \mathbf{1}\{A_r[i,j] > C_{u,v}\}, (u,v) = (\eta_r(i), \eta_r(j)),$$
which is the probability-weighted fraction of circuits where the demand on (u, v) exceeds the current allocation $C_{u,v}$.
3) *Select top candidates.* Sort remote pairs by score, and choose the top-k pairs. Filter by affordability with respect to the remaining budget.
4) *Commit and update.* Assign one EP to each selected pair, update C, and refresh the optimal qubit allocation function η_r.
5) *Iteration.* Repeat Steps 2–4 until the budget is exhausted.

Algorithm 3. Subgradient Descent EP Allocation (SGD-C). The `SGD-C` algorithm formulates the allocation as a continuous optimization problem and applies the projected subgradient descent. It systematically exploits workload information to guide allocations. The steps are as follows:

1) *Initialization.* Start with $C = 0$ and a step size α. Compute the optimal qubit allocation function η_r for each training circuit A_r.
2) *Subgradient update.* For each memory pair (u, v), compute the subgradient using Danskin's theorem[2]:
$$G_{u,v} = -B_{u,v} \sum_r p_r \cdot \mathbf{1}\{A_r[i,j] > C_{u,v}\}, (u,v) = (\eta_r(i), \eta_r(j)).$$
3) *Projection.* Update $C \leftarrow C - \alpha G$ and project back onto the feasible budget set $\sum_{u<v} C_{u,v}\, B_{u,v} \le K$. Refresh the optimal qubit allocation function η_r.
4) *Iteration and quantization.* Repeat Steps 2–3 until convergence, then round C to integers and apply a top-up step to spend the remaining budget.

VII. Qubit Allocation under Pre-Distributed EPs

Having determined the allocation matrix C of pre-distributed EPs across the network, we next address the problem of *qubit allocation*: mapping the qubits of a given circuit onto the network memories. The allocation must jointly exploit the pre-distributed EPs in C and the structure of the circuit so as to minimize the overall execution cost.

Qubit-Allocation Problem Formulation. For a given circuit adjacency matrix A, network-coupling matrix B, and EP allocation matrix C, a qubit-allocation function $\eta : Q(C) \to \bigcup_i M(P_i)$ assigns each circuit qubit to a distinct network memory. The execution cost under η is
$$J(A, C, \eta) \;=\; 2 \sum_{i<j} \max\big(A_{ij} - C_{\eta(i),\eta(j)},\, 0\big)\; B_{\eta(i),\eta(j)}.$$
The qubit-allocation problem is to determine an assignment $\eta^\star$ that minimizes this cost:
$$\eta^\star \;=\; \arg\min_\eta J(A, C, \eta).$$
When $C \equiv 0$, the problem reduces to the classical minimum quadratic assignment problem (min-`QAP`) [19], which is NP-Hard. Thus, the qubit-allocation problem remains computationally intractable. In the following, we present three algorithms to solve the qubit-allocation problem: `Greedy-`η, `exQAP-`η, and `Annealing-`η. They represent complementary strategies for the intractable quadratic assignment:

[1]For each training run of C, we fix a single qubit allocation algorithm (e.g., `Greedy-`η) to serve as the η oracle. The algorithms to determine η are described in Sec. VII.

[2]Danskin's theorem states that if $V(x) = \min_y f(x, y)$, then a subgradient of V with respect to x is given by the partial derivative of $f(x, y)$ with respect to x, evaluated at any minimizer $y^*(x)$. In our case, $x = C$, $y = \eta$, and $f(C, \eta) = J(A, C, \eta)$.

`Greedy-`η offers a lightweight heuristic with local refinement, `exQAP-`η leverages approximation ideas from the maximum QAP to exploit structural information, and `Annealing-`η uses probabilistic search to escape local minima.

Algorithm 4. Greedy Qubit Allocation (Greedy-η). The `Greedy-`η algorithm ranks qubits by gate load, seeds the first assignment using the $C/(1+B)$ score, then assigns the remaining qubits by a greedy marginal-cost rule, and finally applies a 2-opt swap-based local search. The steps are as follows:

1) *Qubit Ordering.* Order qubits by the total gate load (row sum of A), in non-increasing order: $q_1, q_2, \ldots, q_n$.
2) *Seeding.* Compute the row score
$$s(u) \;=\; \sum_{v} \frac{C_{u,v}}{1 + B_{u,v}}\,,$$
and set $\eta(q_1) \leftarrow \arg\max_u s(u)$.
3) *Greedy Placement.* For $i = 2, \ldots, n$, assign $\eta(q_i)$ to an unassigned memory $\bar{m}$ that minimizes the current objective increment
$$\Delta J_i(\bar{m}) \;=\; \sum_{j \in \{1,\ldots,i-1\}} \max\big(A_{ij} - C_{\bar{m},\eta(j)},\, 0\big)\; B_{\bar{m},\eta(j)}.$$
4) *2-opt Local Search.* Repeatedly consider swaps of two qubits (p, q) and accept the first swap that yields a negative cost change $\Delta J_{p,q} = J(A, C, \eta^{\text{swap}(p,q)}) - J(A, C, \eta)$, until no improving swap exists.

Algorithm 5. Extended Quadratic Assignment (exQAP-η). The min-`QAP` problem is inapproximable within any constant factor [19] (unless P=NP). In contrast, the max-`QAP` admits a 4-approximation when one graph satisfies the triangle inequality [20]. Our approach adapts this idea by incorporating the pre-distributed EPs into the objective, transforming the allocation into an extended max-`QAP` instance that can be tackled with assignment and local refinement. The steps are as follows:

1) *Initialization.* Following the AHS heuristic[3], compute qubit gate loads $g(i) = \sum_j A_{ij}$. Compute a greedy max-weight matching M on B and set
$$b(u) = \begin{cases} B_{u,\pi(u)}, & \text{if } (u, \pi(u)) \in M, \\ 0, & \text{otherwise,} \end{cases}$$
then form residual scores
$$s(u) = \sum_{v} B_{uv}\, \max(\bar{a} - C_{uv}, 0), \quad \bar{a} = \tfrac{1}{n(n-1)} \sum_{i \neq j} A_{ij}.$$
Construct $F \in \mathbb{R}^{m \times n}$ with $F_{u,i} = b(u)\, g(i) + s(u)$ and solve a Hungarian assignment on F to obtain the initial η.

[3]Arkin–Hassin–Sviridenko (AHS) proposed an assignment-seeding method for max-QAP that combines structure and load information via a surrogate cost matrix, then solves it with Hungarian [20]. We adapt their idea to incorporate pre-distributed EPs through C.

2) *Iterative Hungarian Refinement.* Given η, form
$$G_{i,u} = 2 \sum_{j} \max(A_{ij} - C_{u,\eta(j)}, 0)\, B_{u,\eta(j)}.$$
Solve Hungarian on G to get η'. If $J(A, C, \eta') < J(A, C, \eta)$, set $\eta \leftarrow \eta'$ and repeat; otherwise stop.
3) *2-opt Local Search.* Apply first-improvement 2-opt swaps, accepting any (p, q) with
$$\Delta J_{p,q} = J(A, C, \eta^{\text{swap}(p,q)}) - J(A, C, \eta) < 0,$$
until no improving swap exists.

Algorithm 6. Simulated Annealing (Annealing-η). Simulated annealing refines qubit allocations by probabilistically accepting suboptimal swaps to escape local minima, balancing exploration and exploitation. The steps are as follows:

1) *Initialization.* Start with seed allocation (e.g., `Greedy-`η or random). Set the temperature $T = T_0$ and the cooling rate $\alpha \in (0, 1)$.
2) *Annealing Loop.* For a fixed number of iterations per temperature:
   - Pick two qubits (p, q) uniformly at random; let η' be η with their memories swapped.
   - Compute $\Delta J = J(A, C, \eta') - J(A, C, \eta)$.
   - If $\Delta J < 0$ accept; else accept with probability $\exp(-\Delta J/T)$.

   After each temperature stage, update $T \leftarrow \alpha T$ and continue until T is small or the iteration budget is reached.
3) *Best-so-far.* Return the best allocation encountered during the run.

VIII. Generalization to Cost-Aware Consumption of Pre-distributed EPs

Motivation. Pre-distributed EPs are shared latency-critical resources. In a multi-tenant quantum network, the operator may wish to discourage excessive consumption of pre-distributed EPs, reserving them for situations where on-demand generation would be particularly costly. In practice, each pre-distributed EP may incur a *usage charge* based on its network distance or its freshness (e.g., remaining lifetime). This motivates extending our formulation to account for the *cost of consuming pre-distributed EPs*, in addition to minimizing the execution latency.

Price Model. We introduce a nonnegative *price matrix* $\Pi = (\pi_{u,v})$ in memories, where $\pi_{u,v}$ denotes the usage charge per-EP in pair (u, v). A natural parameterization is:
$$\pi_{u,v} = \beta_d\, B_{u,v} + \beta_f\, \varphi_{u,v},$$
where B is the network distance matrix, $\varphi_{u,v} \in [0, 1]$ is a freshness score, and $\beta_d, \beta_f \geq 0$ are operator-defined weights.

Charge Constraint. For a circuit adjacency matrix A and mapping η, the total consumption of pre-distributed EPs is
$$U(A, C, \eta) \;=\; \sum_{i<j} \min\big(A_{ij},\, C_{\eta(i),\eta(j)}\big),$$

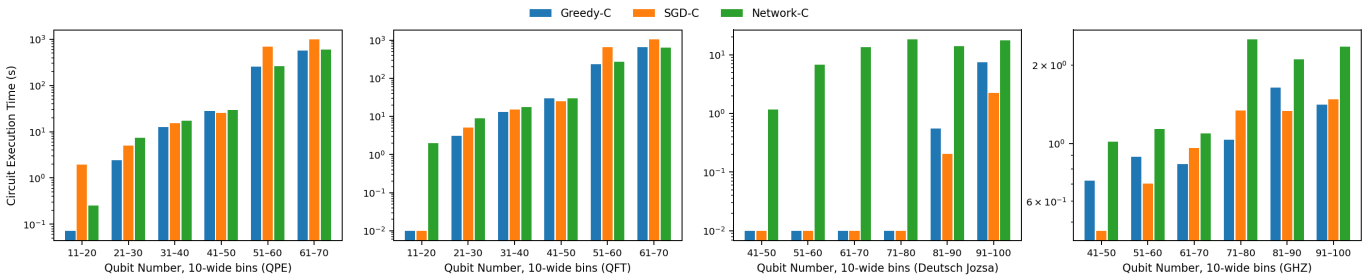

Fig. 2. Execution time of four benchmark families (QPE, QFT, Deutsch Jozsa, GHZ) under three pre-distributed EP trainers (`Greedy-C`, `SGD-C`, `Network-C`). Qubit allocation is fixed to `Greedy-`η. Results are shown for 10-wide qubit bins; the y-axis is logarithmic.

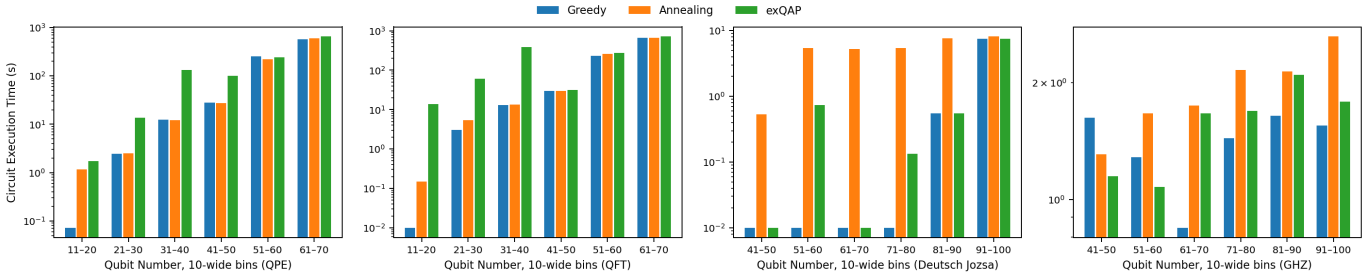

Fig. 3. Execution time of four benchmark families under three qubit-allocation algorithms (`Greedy-`η, `exQAP-`η, `Annealing-`η). The pre-distributed EP trainer is fixed to `Greedy-C`. Results are shown for 10-wide qubit bins; the y-axis is logarithmic.

with an associated usage charge

$$\Gamma(A, C, \eta; \Pi) \;=\; \sum_{i<j} \pi_{\eta(i),\eta(j)} \cdot \min\big(A_{ij},\, C_{\eta(i),\eta(j)}\big).$$

Two Formulations. We consider two natural ways to incorporate usage charges:

- *Hard-cap:* minimize execution latency subject to a usage budget:

$$\min_{\eta} \; J(A, C, \eta) \quad \text{s.t.} \quad \Gamma(A, C, \eta; \Pi) \le \Phi.$$

- *Soft-tradeoff:* include charges directly in the objective:

$$\min_{\eta} \; J(A, C, \eta) \;+\; \lambda\, \Gamma(A, C, \eta; \Pi),$$

  where $\lambda \ge 0$ is a tunable tradeoff parameter.

Simplification. Using $\min(x, y) = x - \max(x - y, 0)$, the soft objective can be rewritten as:

$$\begin{aligned} &J(A, C, \eta) + \lambda \Gamma(A, C, \eta; \Pi) \\ =&\sum_{i<j} \big(B_{\eta(i),\eta(j)} - \lambda \pi_{\eta(i),\eta(j)}\big) \cdot \max(A_{ij} - C_{\eta(i),\eta(j)}, 0) + \text{const}, \end{aligned}$$

where the constant term is independent of η. Thus, optimizing over η is equivalent to solving the original problem with a *shifted distance matrix*:

$$B' \;=\; B \;-\; \lambda \Pi.$$

This substitution allows all of our existing η-methods (`Greedy-`η, `exQAP-`η, `Annealing-`η) to be applied without modification. For the hard-cap formulation, a Lagrangian argument yields the same structure, and a suitable λ can be found by tuning until the budget Φ is met.

Discussion. This model enables fine-grained control over how aggressively pre-distributed EPs are consumed. By assigning higher prices to fresher or longer-distance pairs, the optimizer is incentivized to reserve such EPs and instead rely on on-demand generation when feasible. Importantly, this cost-awareness layer is lightweight: it requires only a shift of the distance matrix before invoking any of the existing qubit allocation algorithms.

IX. Evaluation

Setup and Goals. We evaluate our approach in *NetSquid*, a quantum-network simulator, using a telegate-based execution model [10] for remote operations and the network/physics layer from Secs. VI–VII. The pricing knob for pre-distributed usage is disabled ($\lambda = 0$ in Sec. VIII), so the primary objective is to minimize execution time. We report results aggregated over benchmark circuits and over randomized network topologies, using common seeds across methods to enable paired comparisons.

Benchmark Circuits. We evaluate four benchmark families from the Munich Quantum Toolkit [21]: Quantum Phase Estimation (QPE), Quantum Fourier Transform (QFT), Deutsch Jozsa, and GHZ-state generation (GHZ). Each family contains circuits of multiple sizes, which we group into inclusive qubit-number bins (e.g., $[11, 20]$, $[21, 30]$, $\ldots$). For each bin, all circuits are used to train the pre-distributed EPs allocation C. To assess performance, we then randomly sample 100 circuits from the same bin as test instances. Each sampled circuit is executed with the trained C and the chosen qubit-allocation method η, and their execution times are summed together. The identical set of 100 test circuits is used across all methods to enable paired comparisons, and when C is trained, the inner placement oracle $\min_\eta$ is instantiated with the *same* η-method

used at the test time (oracle-aligned training). This evaluation protocol allows us to compare fairly among C-trainers, η-methods, and their joint combinations.

Generating Random Networks. The network comprises 10 compute nodes, each with 10 quantum memories. We use a network spread over an area of $100km \times 100km$[4]. We use the Waxman model [22] to create Internet topologies, distribute nodes, and create links[5].

Network Parameters. We use values of network parameters similar to those used in [23]. In particular, we set the atomic-BSM probability of success and latency to be 0.4 and 10μ seconds and the optical-BSM probability of success to be 0.3. We use atom-photon generation times and probability of success as 50μ sec and 0.33, and the decoherence threshold of 1 second.

Budget K. For each randomized network instance, we set the pre-distribution budget K as

$$K \;=\; \frac{N}{2} \cdot \overline{d}_{\text{remote}},$$

where N is the total number of memories in the network and $\overline{d}_{\text{remote}}$ is the average shortest-path distance computed over unordered memory pairs that lie on different compute nodes. This fixes a concrete, reproducible budget per network instance and scales with the network size and geometry.

Algorithms Compared. We evaluate three types of comparison.

1) We compare the three algorithms for training pre-distributed EPs from Sec. VI: `Greedy-C`, `SGD-C`, and `Network-C`, while fixing the qubit allocation to `Greedy-`η (Fig. 2).
2) We compare the three algorithms for the allocation of qubits in Sect. VII: `Greedy-`η, `Annealing-`η and `exQAP-`η—while fixing C to be trained by `Greedy-C` (Fig. 3).
3) We evaluate *oracle-aligned combinations*, where C is trained using the same η-method that is applied at the test time, thus capturing the joint effect of C-training and η-allocation. In particular, we consider the 2×2 set of combinations obtained from $\{$`Greedy-C`, `SGD-C`$\}$ crossed with $\{$`Greedy-`η, `Annealing-`$\eta\}$ (Fig. 4).

Evaluation Results. We evaluate the algorithms over the circuits and networks described above. Note that the y-axis is logarithmic and tiny values are clipped at the lower bound of the axis. We observe the following:

1) In the smallest QFT bin and the first four DJ bins in Fig. 2, `Greedy-C`, and `SGD-C` cover essentially all remote gates within the budget K, yielding near-zero execution time (clipped at the lower bound of the log scale). In other figures there are similar phenomena.
2) In Fig. 2, `Greedy-C` performs best on QPE and QFT, while `SGD-C` is stronger on Deutsch Jozsa; GHZ shows mixed behavior. The structure-only `Network-C` trainer underperforms the circuit-aware trainers.
3) In Fig. 3, `Greedy-`η performs best among the three algorithms for qubit allocation.
4) In Fig. 4, it shows a trend similar to that in Fig. 2. `Greedy-C` +`Greedy-`η performs best in QPE and QFT, while `SGD-C` +`Greedy-`η is stronger in Deutsch Jozsa; GHZ shows mixed behavior.

Why Benchmark Families Differ. The circuit families induce different two-qubit demand profiles. QFT and QPE involve dense long-range couplings, so even small misplacements in η leave substantial uncovered demand, and execution time grows rapidly with n. Deutsch–Jozsa has a much fewer effective demand; with a moderate budget K, most remote gates can be eliminated, keeping times close to the log-scale floor and reducing differences between methods. GHZ circuits form a chain-like pattern whose cost depends strongly on where the chain anchors in the network; as a result, assignment methods trade off differently, producing mixed winners across bins.

Takeaway. Circuit-aware training of pre-distributed EPs (`Greedy-C`, `SGD-C`) together with effective qubit placement (`Greedy-`η) consistently achieves the lowest execution times. This underscores that optimizing both EP allocation and qubit placement in tandem is crucial to minimize execution latency in distributed quantum computation.

X. Conclusions

In this work, we addressed the `DQC-PD` problem: executing a quantum circuit across a network with the aid of pre-distributed entanglement pairs. We developed and evaluated algorithms for both pre-distributed EP allocation and qubit-to-memory assignment, showing that their joint optimization is crucial for minimizing execution latency. Looking ahead, several extensions are natural. One is to incorporate operations based on teleportation and cat-entanglement into our framework. Another is to extend our algorithms to the online setting, where circuits arrive dynamically and EPs must be replenished intelligently. Addressing these challenges will further advance the goal of scalable and efficient distributed quantum computation.

[4]The $100km \times 100km$ area is a modeling choice to set a representative spread of inter-QPU distances; under uniform rescaling of all link lengths, the cost matrix B (and budget K if scaled accordingly) rescales by a constant factor, so the allocation decisions are driven primarily by relative inter-QPU costs rather than the absolute geographic scale.

[5]Inter-QPU distances parameterize B (inter-node EP cost), while intra-QPU connectivity affects local compilation overhead (e.g., swap operations).

References

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

[2] R. G Sundaram, H. Gupta, and C. Ramakrishnan, "Efficient distribution of quantum circuits," in *35th International Symposium on Distributed Computing (DISC 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[3] M. Zomorodi-Moghadam, M. Houshmand, and M. Houshmand, "Optimizing teleportation cost in distributed quantum circuits," *International Journal of Theoretical Physics*, vol. 57, pp. 848–861, 2018.

[4] R. G. Sundaram and H. Gupta, "Distributing quantum circuits using teleportations," in *2023 IEEE International Conference on Quantum Software (QSW)*. IEEE, 2023, pp. 186–192.

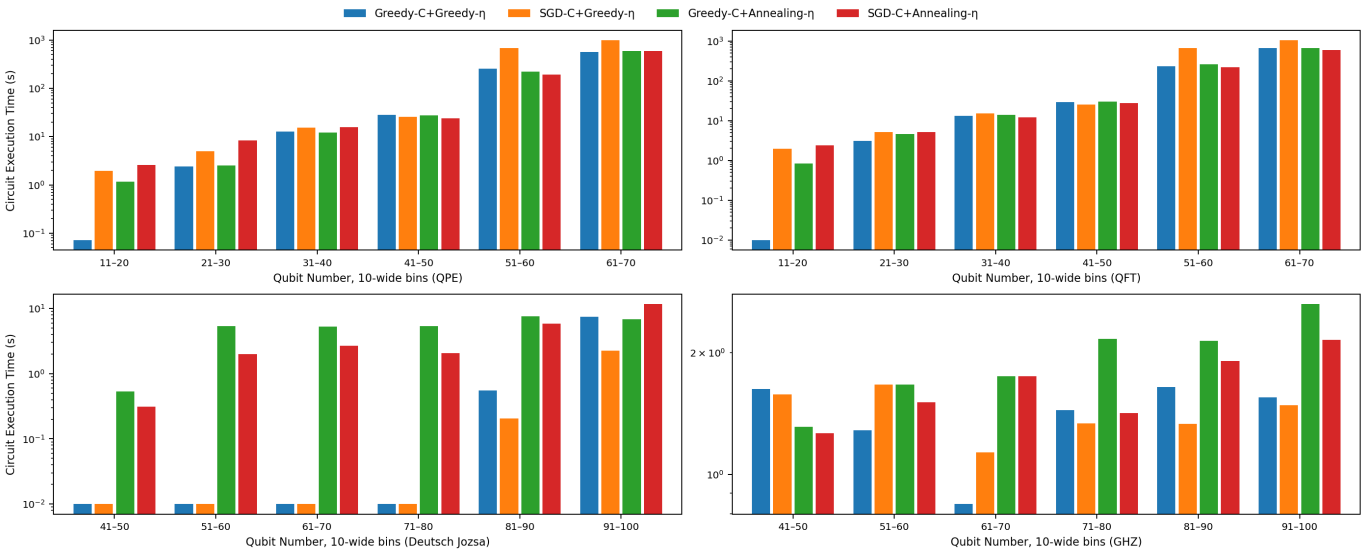

Fig. 4. Execution time of four benchmark families under joint combinations of EP trainers and qubit-allocation methods. We report oracle-aligned training, i.e., C is trained using the same η-method applied at test time. Results are shown for 10-wide qubit bins; the y-axis is logarithmic.

[5] E. Nikahd, N. Mohammadzadeh, M. Sedighi, and M. S. Zamani, "Automated window-based partitioning of quantum circuits," *Physica Scripta*, vol. 96, no. 3, p. 035102, 2021.

[6] D. Cuomo, M. Caleffi, K. Krsulich, F. Tramonto, G. Agliardi, E. Prati, and A. S. Cacciapuoti, "Optimized compiler for distributed quantum computing," *ACM Transactions on Quantum Computing*, vol. 4, no. 2, feb 2023.

[7] D. Ferrari, A. S. Cacciapuoti, M. Amoretti, and M. Caleffi, "Compiler design for distributed quantum computing," *IEEE Transactions on Quantum Engineering*, vol. 2, pp. 1–20, 2021.

[8] Y. Yang, R. G. Sundaram, and H. Gupta, "Efficient execution of multiple quantum circuits over a quantum network," in *2025 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2025.

[9] R. G Sundaram, H. Gupta, and C. Ramakrishnan, "Distributed quantum computation with minimum circuit execution time over quantum networks," in *IEEE QCE*, 2024.

[10] R. G. Sundaram and H. Gupta, "Dynamic distribution of quantum circuits with minimum execution time," in *2025 International Conference on Quantum Communications, Networking, and Computing (QCNC)*. IEEE, 2025.

[11] X. Fan, C. Zhan, H. Gupta, and C. R. Ramakrishnan, "Optimized distribution of entanglement graph states in quantum networks," *IEEE Transactions on Quantum Engineering*, vol. 4, pp. 1–16, 2023.

[12] X. Fan, Y. Yang, H. Gupta, and C. R. Ramakrishnan, "Distribution and purification of entanglement states in quantum networks," in *Proceedings of the IEEE International Conference on Quantum Communications and Networking (QCNC)*, 2025.

[13] C. H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W. K. Wootters, "Teleporting an unknown quantum state via dual classical and einstein-podolsky-rosen channels," *Physical review letters*, vol. 70, no. 13, p. 1895, 1993.

[14] O. Daei, K. Navi, and M. Zomorodi-Moghadam, "Optimized quantum circuit partitioning," *International Journal of Theoretical Physics*, vol. 59, no. 12, pp. 3804–3820, 2020.

[15] R. G. Sundaram, H. Gupta, and C. Ramakrishnan, "Distribution of quantum circuits over general quantum networks," in *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2022, pp. 415–425.

[16] Y. Mao, Y. Liu, and Y. Yang, "Probability-aware qubit-to-processor mapping in distributed quantum computing," in *Proceedings of the 1st Workshop on Quantum Networks and Distributed Quantum Computing*, 2023, pp. 51–56.

[17] ——, "Qubit allocation for distributed quantum computing," in *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*. IEEE, 2023, pp. 1–10.

[18] M. Ghaderibaneh, H. Gupta, C. R. Ramakrishnan, and E. Luo, "Predistribution of entanglements in quantum networks," in *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*, 2022, pp. 426–436.

[19] S. Sahni and T. Gonzalez, "P-complete approximation problems," *Journal of the ACM (JACM)*, vol. 23, no. 3, pp. 555–565, 1976.

[20] E. M. Arkin, R. Hassin, and M. Sviridenko, "Approximating the maximum quadratic assignment problem," *Information Processing Letters*, vol. 77, no. 1, pp. 13–16, 2001.

[21] N. Quetschlich *et al.*, "Mqt bench: Benchmarking software and design automation tools for quantum computing," *Quantum*, 2023.

[22] B. M. Waxman, "Routing of multipoint connections," *IEEE journal on selected areas in communications*, vol. 6, no. 9, pp. 1617–1622, 1988.

[23] M. Ghaderibaneh, C. Zhan, H. Gupta, and C. R. Ramakrishnan, "Efficient quantum network communication using optimized entanglement swapping trees," *IEEE Transactions on Quantum Engineering*, vol. 3, pp. 1–20, 2022.