Distributed Quantum Computation with Minimum Circuit Execution Time over Quantum Networks

Ranjani G. Sundaram
Stony Brook University, NY

Himanshu Gupta
Stony Brook University, NY

C. R. Ramakrishnan
Stony Brook University, NY

***Abstract*—Present quantum computers are constrained by limited qubit capacity and restricted physical connectivity, leading to challenges in large-scale quantum computations. Distributing quantum computations across a network of quantum computers is a promising way to circumvent these challenges and facilitate large quantum computations. However, distributed quantum computations require entanglements (to execute remote gates) which can incur significant generation latency and, thus, lead to decoherence of qubits. In this work, we consider the problem of distributing quantum circuits across a quantum network to minimize the execution time. The problem entails mapping the circuit qubits to network memories, including within each computer since limited connectivity within computers can affect the circuit execution time.**

We provide two-step solutions for the above problem: In the first step, we allocate qubits to memories to minimize the estimated execution time; for this step, we design an efficient algorithm based on an approximation algorithm for the max-quadratic-assignment problem. In the second step, we determine an efficient execution scheme, including generating required entanglements with minimum latency under the network resource and decoherence constraints; for this step, we develop two algorithms with appropriate performance guarantees under certain settings or assumptions. We consider multiple protocols for executing remote gates, viz., telegates and cat-entanglements. With extensive simulations over NetSquid, a quantum network simulator, we demonstrate the effectiveness of our developed techniques and show that they outperform a scheme based on prior work by up to 95%.

I. Introduction

Present quantum computers have limited qubit capacity and are susceptible to noise due to decoherence and gate operations. Error-correcting codes can be used to overcome noisy gate operations, but that results in a blowup in the number of qubits, which further exacerbates the qubit capacity hurdle. Distributing quantum computations over a network of quantum computers (QCs) is a promising way to facilitate large computations over current QCs. However, distributed quantum computations require entanglements (to execute remote gates) which can incur significant generation latency and, thus, lead to decoherence of qubits. Thus, in this work, we consider the problem of distributing quantum circuits across a quantum network with the optimization objective of minimizing the circuit execution time, which includes latency incurred in generating the required entanglements to execute the remote gates under decoherence constraints.

Distributing quantum circuits entails mapping the circuit's qubits to the quantum network's qubit memories and introducing quantum communication operations to execute remote gates (gates spanning multiple QCs). Thus, in our proposed approaches, we first determine an efficient allocation of qubits to memories in the computers such that the estimated estimated execution time is minimized, and after having established the mapping of qubits, we use efficient strategies to execute the gates using required entanglements with minimum latency under network resource and decoherence constraints. The allocation of qubits to memories also entails mapping the qubits within each computer since limited connectivity within computers can affect the circuit execution time.

This work was partly supported by the National Science Foundation under Award FET-2106447 and Award CNS-2128187.

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. However, almost all works on distributing quantum circuits have focused on the objective of minimizing the number of maximally-entangled pairs (EPs) either by minimizing the number of cat-entanglements [1, 13] or the number of teleportations [19, 24, 28]. In the work closest to ours, given an allocation of qubits to computers, [8] minimizes the number of time slots to generate EPs required to execute the remote gates; they make the simplistic assumption that each EP's generation takes a single unit of time. In our work, we aim to comprehensively solve the problem of the distribution of quantum circuits by considering the optimization objective of minimizing the circuit execution time under the network resource and decoherence constraints. *The circuit execution time must include generation latencies of the required EPs, which must take into consideration the stochasticity of the underlying processes; this makes the problem particularly challenging and significantly different from prior works on distributing quantum circuits.* When allocating the qubits to network memories, we also map qubits to memories *within each computer* as limited connectivity within computers also affects the circuit execution time.

Our Contributions. In this paper, we formulate the problem of distributing quantum circuits in quantum networks to minimize circuit execution time under given constraints. We address this problem in two steps, as below.

- For the first step of allocating circuit qubits to network memories, we develop a heuristic scheme based on an approximation algorithm for a special case of the well-known maximum quadratic assignment problem. (§V).
- For the second step of developing an efficient execution scheme for a given qubit allocation, we design the following algorithms.
  - For the special case, when the consumption order of the required EPs is total, we develop a provably optimal dynamic programming approach; we generalize it to the general case.

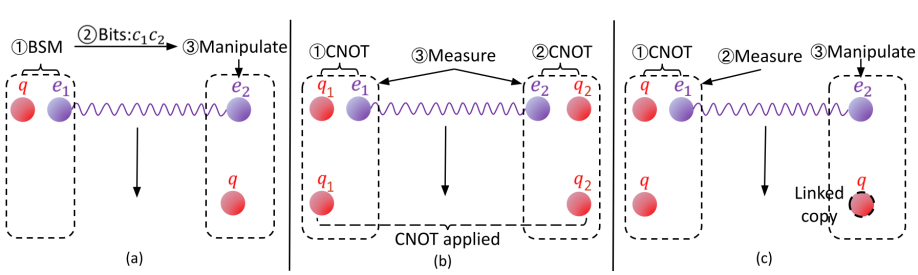

Fig. 1: Quantum communication. (a) Teleportation, (b) Telegate, (c) Cat-Entanglement. Dashed boxes are the network nodes; the initial (final) state of the qubits is at the top (bottom).

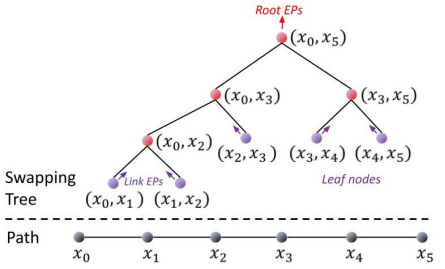

Fig. 2: Swapping tree to generate remote EPs.

- For the special case, when the consumption order of the required EPs is null, we develop a greedy heuristic with appropriate performance guarantees under reasonable assumptions; we generalize it to the general case.
- When using cat-entanglements to execute remote gates, we develop a scheme for selecting a provably near-optimal set of cat-entanglements. (§VII)

- Finally, we demonstrate the effectiveness of our developed techniques by evaluating them on randomly generated circuits and known benchmarks and show that our techniques outperform the prior work by up to 95%. (§IX)

II. Background

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 either 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.

A. Distribution of Quantum Circuits over QNs

Given a quantum network (QN) and a quantum circuit, we seek to distribute and execute the given circuit over the network in 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.

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 qubit is subject to unrecoverable errors, we consider the following ways to communicate qubits across network nodes.

Teleportation. An alternative approach to physically transmit a qubit from a node A to B is via *teleportation* [3] which requires an a priori distribution of an EP over A and B. See Fig. 1(a). Teleportation can be used to bring operands of a remote gate to a single QC for local execution.

Telegates. One way to execute remote gates without communicating the gate operands are via the telegate protocol [8]. The telegate protocol (see Fig. 1(b)) is a sequence of local operations that effectuates the execution of a remote $CNOT$ gate with operands at nodes A and B; the protocol requires an a priori distribution of an EP over A and B.

Cat-Entanglement: Creating "Linked Copies" of a Qubit. Yet another approach to execute remote gates is by creating *linked read-only copies* of a qubit across QCs via *cat-entanglement* operations [11, 27]. Creating a linked copy of a qubit q at node A to a node B requires a priori distribution of an EP over A and B. See Fig. 1(c). The linked copy at B can be used to execute binary gates where q is the (read-only) control qubit, until a unary operation needs to be executed on q. When a unary operation needs to be executed on q, a disentanglement operation is done, which destroys the linked copies, and then, the unary operation can be done on the original qubit q at A.

Teleportation vs. Telegates vs. Cat-Entanglements. First, observe that each of these three communication protocols requires a single EP. Teleportation transports the qubit operand to the computer where the gate is executed. In contrast, the telegate protocol executes a remote gate without moving qubits. The main advantage of cat-entanglement is that one cat-entanglement (and, thus, one linked copy) can sometimes be used to execute several binary gates; however, cat-entanglements are best used for circuits with only *CZ* binary gates (see §VII). This paper considers both telegates and cat-entanglements (§VII) for executing remote gates. The use of teleportations leads to dynamic qubit allocation (see §III), which requires more a sophisticated problem formulation and solution, and is thus deferred to our future work.

B. Generating EPs over Remote Nodes

Each of the above communication modes requires an apriori distributed EP. One simple way to have an EP distributed over nodes A and B is to generate an EP locally (at some node) and transport the qubits to A and B respectively; however, this involves direct communication of qubits, which may not be feasible due to irrecoverable errors over large distances. To circumvent this challenge, to distribute an EP over remote nodes, we generate EPs over larger and larger distances from shorter-distance EPs using a sequence of "entanglement-swapping" operations. An entanglement swapping (ES) operation can be looked up as being performed over three nodes (A, B, C) with two EPs over (A, B) and (B, C); the ES operation results in an EP over the nodes (A, C), by essentially teleporting the

first qubit in node B (i.e., the qubit of the EP over (A, B)) to the node C using the second EP over (B, C). In general, an EP over a pair of remote nodes A and B can be generated by a sequence of ES operations over the EPs over adjacent nodes along a path from A to B. For example, see Fig. 2 which shows a "swapping" tree to generate an EP over (x_0, x_5) using EPs along the path $x_0, x_1, x_2, x_3, x_4, x_5$. Different swapping trees over the same path P can incur different generation latencies [14].

Generation Latency of EPs. The stochastic nature of the ES operations means that an EP at the swapping tree's root is successfully generated only after many failed attempts; thus the generation of an EP incurs significant *generation latency*. To generate a set of EPs concurrently, we need to allocate the network resources appropriately to each EP's generation.

C. Decoherence and Fidelity

Fidelity is a measure of how close a realized state is to the ideal. The fidelity of a quantum state decreases over time due to interaction with the environment and gate operations. Time-driven fidelity degradation is called *decoherence*. To bound decoherence, we limit the aggregate time a qubit spends in a quantum memory before being consumed by the *decoherence threshold* of τ seconds. We address fidelity degradation by using entanglement purification [4] techniques which entails using multiple copies of an EPs to improve its fidelity.

III. Problem Formulation and Related Work

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

Qubit-Allocation Function. Let the set of network nodes/QCs in the given quantum network be $\mathbb{P} = \{P_1, P_2, \ldots, P_N\}$, where each QC P_i is equipped with a set $Q(P_i)$ of qubit memories for data storage. For a given quantum circuit C, let $Q(C)$ be the set of qubits in C. To execute a circuit over a quantum network, we must first distribute the circuit qubits over the qubit memories in the network nodes. To that end, we define the *qubit allocation function* η as the function that maps the circuit qubits $Q(C)$ to the network qubits $\bigcup_i Q(P_i)$, i.e., $\eta : Q(C) \rightarrow \bigcup_i Q(P_i)$; the qubit-allocation function must be one-to-one, i.e., only one circuit-qubit can be mapped to a qubit memory in the network.

Remote and Local Gates. For a given circuit C and a qubit allocation function η, a gate (q_i, q_j, t) in C is considered to be *remote* if q_i and q_j are assigned to different *computers* by the qubit allocation function, i.e., $\eta(q_i) \in Q(P_i)$ and $\eta(q_j) \in Q(P_j)$ where $i \neq j$. Every gate that is *not* a remote gate is considered a *local gate*.

Coupling Graph; Executing Local Gates Using Swaps. In realistic quantum computer hardware, quantum gates can be executed only over those qubit operands that are in memories located "close by." To characterize this limitation, we use a concept of *coupling graph* U_i over the set of qubits, $Q(P_i)$ for each computer P_i; an edge (m_a, m_b) in U_i signifies that the memories m_a and m_b are located close by that qubits in them can be operated upon directly using a binary gate. If two qubits q_c and q_d are in far away memories m_c and m_d within a computer P_i, then to execute a gate (q_c, q_d, t), we need to move the qubits close-by through a series of swap operations. In particular, we assume that the swaps are done along the shortest path connecting m_c and m_d, and thus, number of swaps needed is equal to the shortest path length between m_c and m_d in the coupling graph U_i.

Enforcing Static Qubit-Allocation Function. In general, the qubit-allocation function may change during the circuit execution due to teleportations and/or swap operations. However, to simplify algorithm design and analysis, we enforce the qubit allocation to be fixed throughout the circuit execution; we relax this enforcement in our implementation and evaluations (§IX). This enforcement of static qubit allocations entails the following:

1) We don't need to consider teleportations to execute remote gates, as they would incur double the cost than telegates, as the qubits will need to be teleported back and forth. Thus, to execute remote gates, we use telegates or cat-entanglements (§VII).
2) For the execution of local gates that require swaps, we reverse the sequence of swap operations after the gate execution to preserve the qubit allocation function. We relax this in our evaluations.

We now formulate the problem of distributing quantum circuits in quantum networks with minimum circuit execution time under given constraints.

Distributed Quantum Computation and Routing (`DQC-QR`) Problem. Given a quantum network and a quantum circuit C, the `DQC-QR` problem is to distribute and execute C over the given network with minimum execution time under the network constraints (in particular, memory at each network node, EP generation resources, and decoherence). The `DQC-QR` problem entails (i) finding a qubit-allocation function, which remains fixed throughout the circuit execution (see above), and (ii) an execution scheme involving the execution of local and remote gates, such that the total circuit execution time is minimized. As discussed above, executing local gates may involve swaps, while executing remote gates requires communication, which requires EPs to be generated, as discussed below.

The `DQC-QR` problem can be shown to be NP-Hard by a reduction from the min quadratic-assignment problem [21].

Key Challenges: Qubit-Allocation, Generation Order of EPs. The key initial task of the `DQC-QR` problem is to determine the qubit-allocation function that facilitates a minimum-time execution scheme. Now, given a qubit-allocation function, the resulting set of remote gates determines the EPs that need to be generated. Without decoherence (and minimal/zero gate and swap latencies), the desired EPs can be generated at the start of a circuit execution in any order. However, ***in the presence of decoherence, the required EPs need to be generated in an appropriate order***—so that each generated

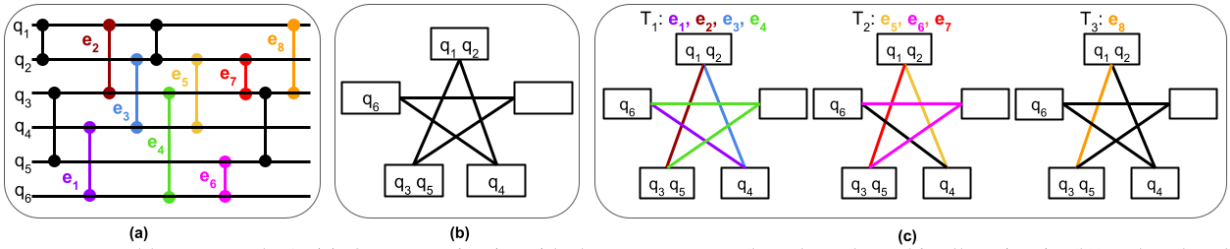

Fig. 3: DQC−QR Problem Example 1. **(a)** Quantum circuit, with the remote gates (based on the qubit allocation in (b)) colored and labeled with the EPs required. **(b)** Qubit allocation over the quantum network. **(c)** Execution scheme, i.e., the batches and order in which the required EPs are generated; the colored paths in (c) represent the entanglement routes along which the corresponding EPs are generated.

EP can be consumed before decohering.[1] In the case of non-zero gate execution and swap latencies, generation of EPs can also be overlapped with swap and gate operations to minimize overall execution time. In summary, the key challenges in solving the DQC−QR problem are to determine an efficient qubit-allocation function and an execution scheme; the latter largely involves determining when and how to generate the required EPs.

Example 1. Fig. 3 shows an instance of the DQC−QR problem, with the input circuit of five qubits in Fig. 3 (a). The colored gates (non-black) represent remote gates based on the qubit allocation over a quantum network shown in Fig. 3(b). Fig. 3(b) doesn't show the coupling graph within each network node, for the sake of clarity, as each node only holds at most two qubits. Fig. 3 (c) shows one possible execution scheme for the given qubit allocation: Execute the EPs in three batches in order: $\{e_1, e_2, e_3, e_4\}, \{e_5, e_6, e_7\}, \{e_8\}$; within each batch, the EPs can be generated concurrently. As EPs get generated, they are consumed to execute the corresponding remote gates.

A. Related Work

The DQC−QR problem is related to two studied problems, viz., Distributing Quantum Circuits and the Qubit Routing. We discuss these below, then discuss the work closest to ours.

Prior Work on Distributing Quantum Circuits (DQC) Problem. The DQC problem is to distribute a given quantum circuit over a quantum network with some optimization objective. In contrast to our DQC−QR problem, the DQC problem ignores the coupling graphs within each computer [1, 9, 13, 19], and thus, the swap operations needed. 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 [17, 18] consider EPs with arbitrary (but fixed) cost and develop a simulated-annealing heuristic for the qubit-allocation part of the DQC problem. Our schemes in this work use a similar two-step strategy as some [13, 24, 25] of the DQC works, but otherwise differ significantly as we need to consider the stochastic and concurrent generation of required EPs to minimize execution time, and the coupling graph; we also consider telegates to execute remote gates while the two-step DQC works consider entanglements [13, 25] and/or teleportations [24, 25].

Prior Work on Qubit Routing (QR) Problem. The QR problem considers limited connectivity across memories within a single QC and entails finding an allocation of circuit qubits to qubit memories, followed by a minimal sequence of swap operations to execute the circuit gates. Most QR works [16, 23, 29] first find an initial allocation of circuit qubits to memories and then add swap gates as needed. The optimization objective is to minimize the number of swap operations or the circuit depth overhead. Some works [5, 7, 22] have used subgraph isomorphism to obtain high-quality qubit allocations followed by token-swapping heuristics to determine a near-optimal set of swap operations. Our work generalizes the QR problem by considering a *network* of QCs, and thus, needs to address the generation of EPs as well as swap operations. We note that our treatment of qubit routing is limited as we keep the qubit allocation fixed (this is relaxed in our evaluations (§IX).

Prior Work on Our DQC−QR Problem. The works closest to ours [8, 12] address the DQC−QR problem with some limitations and differences. In particular, [12] 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. In the closest work that inspires this work, [8] 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 (one time slot), and ignore decoherence constraints. In §IX, we compare our approaches with their approach (supplemented with our qubit-allocation strategy). The main difference between our work and [8] is that we consider qubit-allocation and the stochastic generation of EPs under decoherence and network resource constraints. We also consider cat-entanglements for executing remote gates, which result in significant performance improvement, as demonstrated in our evaluations.

IV. High-Level Approach

In the previous section, we formulated the DQC−QR problem as distributing a given circuit over a given network with minimum execution time. As mentioned above, the key challenges or subproblems in solving the DQC−QR problem is to determine the qubit-allocation function and the execution scheme such that the execution time is minimized. Thus, it is natural to tackle the DQC−QR problem as a sequence of two steps, similar to the approaches taken in prior works on the similar problem of distributing quantum circuits [13, 24, 25].

[1]Note that the order of gates in the given circuit imposes a consumption order on the EPs which, in turn, imposes a generation order on EPs and may also require them to wait before being consumed.

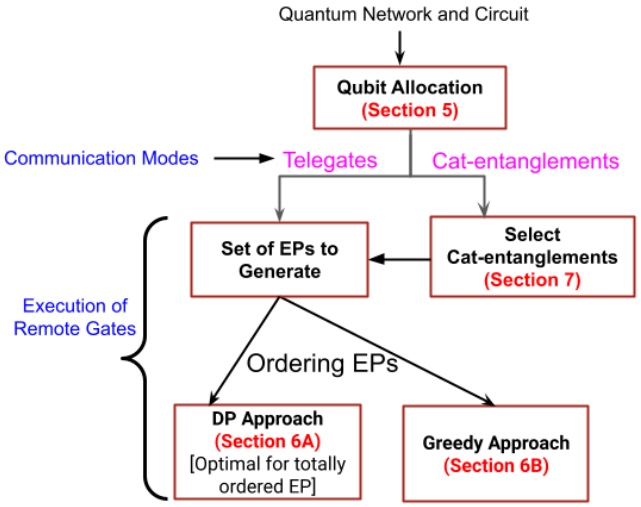

Fig. 4: Overall plan and organization of the schemes developed.

1) *Determine the Qubit-Allocation Function.* In this step, given the quantum circuit and the quantum network, we determine the qubit-allocation function that will yield an execution scheme in the second step with minimum total circuit execution time.
2) *Determine the Execution Scheme, Given the Qubit Allocation.* In this step, given a qubit-allocation function for a given circuit, we determine the execution scheme—which largely entails determining when and how to generate the EPs required to execute the remote gates. Local gates are executed using appropriate swap operations.

We discuss the above steps in the following sections. See Fig. 4 for the overall plan and organization of the following sections.

V. Step 1: Determining Qubit Allocation Function

In this section, we design an algorithm for determining a qubit-allocation function to yield an efficient execution scheme. We start with formally defining the qubit-allocation problem and show that it is NP-Hard to approximate within a constant factor. Then, we design an algorithm for qubit allocation based on a 4-factor approximation algorithm for a special case of the related quadratic-assignment problem.

Qubit-Allocation Problem Formulation. Informally, the qubit-allocation problem essentially maps a circuit graph (defined over circuit qubits, with edge weights representing the number of gates between the qubit-pairs) onto a network-coupling graph (defined over network memories, with edge weights representing the cost of executing a gate over the qubits in the corresponding network memories); the optimization objective is to minimize an appropriately defined cost of the mapping. In effect, we want to map the circuit qubits to network memories so that qubit-pairs with more gates between them are mapped to memories that are "close by"; this can be formally captured by defining a mapping cost as the weighted sum of the product of the weights of the edge pairs that map to each other in the mapping. We formalize the above intuition below by defining the input graphs and the problem.

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

Network-Coupling Graph G_N. The network-coupling graph is an edge-weighted graph defined over the set of qubit-memories $\bigcup_i Q(P_i)$ in the given quantum network with QCs $\{P_i\}$. The weight $w'(m_a, m_b)$ associated with an edge between two memories (m_a, m_b) is defined as follows.

- If m_a and m_b are in the same computer P_i, then the weight is the product of swap-operation latency and twice the shortest distance between m_a and m_b in the coupling graph U_i of P_i; note that this represents the time to execute the gate over qubits in m_a and m_b using swap operations.
- If $m_a \in Q(P_i)$ and $m_b \in Q(P_j)$, then the weight on the edge (m_a, m_b) is the latency of (independently) generating an EP over network nodes P_i and P_j.

For the sake of simplicity, the above weighting of the network-coupling graph assumes that each EP is generated independently. However, when we actually generate the EPs in the following step (Step 2 discussed in §VI), we generate them as concurrently as possible. We now formally define the qubit-allocation problem.

Qubit-Allocation Problem Formulation. Given a quantum circuit C and a quantum network, the qubit-allocation problem is to determine a mapping $\eta : Q(C) \to \bigcup_i Q(P_i)$ from the circuit qubits to the qubit-memories of the given network, such that the mapping-cost $\text{cost}(\eta)$, defined as below, is minimized.

$$\text{cost}(\eta) = \sum_{q_i, q_j \in Q(C)} w(q_i, q_j) \times w'(\eta(q_i), \eta(q_j))$$

The qubit-allocation problem can be shown to be NP-Hard by a reduction from the well-known `minimum quadratic assignment` problem (min-`QAP`) [21]. In addition, by the same reduction, it can also be shown that there is no constant-factor polynomial-time approximation algorithm for the qubit-allocation problem unless P=NP.

Below, we design an algorithm to solve the qubit-allocation problem. We start with a formal definition of the min-QAP problem since our designed algorithms are based on it.

Minimum Quadratic Assignment Problem. Given two $n \times n$ matrices A and B, the min-`QAP` problem is to permute the rows/columns in B such that the sum of the product of the matrix elements is minimized. More formally, the goal is to find a permutation (a one-to-one mapping) $P : \{1, 2, \ldots, n\} \mapsto \{1, 2, \ldots, n\}$ to minimize the following quantity: $\sum_{1 \le i,j \le n} A[i, j]B[P(i), P(j)]$.

Qubit-Allocation Algorithm Based on Quadratic-Assignment. The qubit-allocation problem is quite similar to the minimum quadratic assignment problem (min-`QAP`) defined above, except that the min-`QAP` problem requires the two input matrices/graphs to be complete and of equal sizes. Moreover, the min-`QAP` problem is known to be inapproximable within any polynomial factor [21] (unless P = NP), but the *maximum* quadratic assignment problem (max-`QAP`) has a 4-factor approximation algorithm when

one of the graphs satisfies the triangular inequality [2]. Based on the above, we design an efficient heuristic for our qubit-allocation problem as follows.

Let C be the given quantum circuit with G_C as the corresponding circuit graph. Let G_N be the network-coupling graph of the given quantum network. Without loss of generality, let us assume that the size of G_C (i.e., the number of circuit-qubits $|Q(C)|$) is less than the number of qubit-memories in the network. Our qubit-allocation algorithm is:

(a) Add dummy nodes and edges to G_C to create a new graph G'_C, which is the same size as the network-coupling graph G_N. In particular, we add an appropriate number of dummy nodes to G_C and connect these nodes with edges of zero weight to the original nodes in G_C. We also add edges of weight zero between all pairs of dummy nodes.
(b) Create a new graph G''_C by replacing each edge-weight $w(q_i, q_j)$ with $M - w(q_i, q_j)$, where M is a sufficiently large number (e.g., equal to the total sum of edge-weights in G_C). This change of weights converts the minimizing mapping-cost problem in G'_C to the maximizing mapping-cost problem in G''_C.
(c) Solve the max-QAP over the input graph G''_C and G_N using the 4-approximation algorithm for max-QAP.

Theorem 1: The 4-approximation max-QAP algorithm from [2] returns a $6/p$-approximate solution for the max-QAP problem over G''_C and G_N; here, p is the entanglement-swapping's success probability.

PROOF: *(Sketch)* First, we show that the edge weights in G_N satisfy the triangular inequality within a constant factor. If m_i and m_j are memories in a computer P, then $w'(m_i, m_j) \leq w'(m_i, m_k) + w'(m_k, m_j)$ for any memory m_k in P since $w'(m_i, m_j)$ is the time to perform swaps along the shortest path between m_i and m_j. If m_i and m_j are in different computers P_1 and P_2, then, for any $m_k \in P_3$, we have [14]:

$$w'(m_i, m_j) \leq 3/(2p)(w'(m_i, m_k) + w'(m_k, m_j)),$$

since an EP over (P_1, P_2) can be generated using EPs over (P_2, P_3) and (P_1, P_3). The $3/(2p)$ factor above, when incorporated in the analysis for the 4-approximation result from [2], yields the $4(3/(2p)) = (6/p)$ approximation factor. ∎

We note that our overall qubit-allocation algorithm (i.e., (a)-(c) above) is still a heuristic since the conversion from the min-QAP to max-QAP does not preserve the approximation factor; however, the algorithm performed well empirically (§IX).

VI. Step 2: Executing Remote Gates by Generating EPs

Step 1, discussed in the previous section, determines the qubit allocation function, which, in turn, determines the set of remote gates that need to be executed using quantum communication mechanisms. In this section, we use telegates to execute remote gates (we consider cat-entanglements in §VII); since telegates require an a priori EP distributed over the nodes/QCs that contain the gate operands—for each remote gate, we need to generate an EP over the pair of nodes that contain the gate operands. For simplicity, we assume the swap and gate execution latencies to be zero (i.e., negligible compared to the EP generation latencies); we relax this assumption in §VIII. Under the above assumption, given a qubit allocation, the problem of executing the given circuit with minimum execution time essentially boils down to generating (possibly, concurrently) the required EPs in minimum total latency (under the constraints of consumption order and decoherence thresholds, as discussed below). We start by considering the simpler case when there is no decoherence.

Fig. 5: Consumption order of EPs required in Example 1. A directed edge from e_i to e_j indicates e_i must be consumed before e_j.

Generating EPs When No Decoherence (`GenerateEPs`). Let us use $\mathcal{E}$ to denote the set of EPs required to be generated for a given qubit-allocation function. Without decoherence constraints, Step 2 of our approach boils down to generating the EPs in $\mathcal{E}$ with minimum total latency. This is because if all the EPs in $\mathcal{E}$ can be generated in total time T, then the total circuit can also be executed in T as all the circuit gates can be executed instantly right after all the EPs are generated; the converse is also true, i.e., if the circuit execution time is T, then it implies that the total generation latency of all the EPs in $\mathcal{E}$ is less than T. To generate a set of EPs $\mathcal{E}$ concurrently with minimal total latency without decoherence, we use the linear programming (LP) approach from [15] that generates a set of desired EPs using swapping trees with a maximum total generation rate. In our context, since we are interested in only generating a *specific set* of EPs with a minimum total *latency*, we modify the LP appropriately after each EP has been generated successfully. We refer to the above procedure to generate a set of EPs concurrently as `GenerateEPs`.

Generating EPs under Decoherence Constraints. If there is decoherence, EPs can decohere while and after (as they wait to be consumed) being generated. In particular, decoherence may require that we partition $\mathcal{E}$ into subsets of EPs (called *batches*) with each batch generated independently—as generation of the entire set $\mathcal{E}$ concurrently may result in some EPs having a generation latency of more than the decoherence threshold τ. Once the set $\mathcal{E}$ has been partitioned appropriately into batches, the batches need to be generated (with each batch generated independently) in an appropriate order to prevent EPs from decohering. This is because if we generate the batches in an arbitrary order, then some already-generated EPs may have to wait for other EPs to be generated and consumed before being consumed, leading to the waiting EPs possibly getting decohered. For example, for the example in Fig. 3, consider two batches of EPs: $\mathcal{E}_1 = \{e_1, e_2, e_5\}$ and $\mathcal{E}_2 = \{e_3, e_4, e_6\}$. Based on the consumption order (formally defined below) in Fig. 5, if $\mathcal{E}_1$ is generated before $\mathcal{E}_2$ then $e_5 \in \mathcal{E}_1$ will need to wait for $e_3 \in \mathcal{E}_2$ to be generated and consumed, before e_5 can be consumed. In this case, even generating $\mathcal{E}_2$ before $\mathcal{E}_1$ will lead to an EP waiting for another. In general, partitioning $\mathcal{E}$ into batches and the order in which these batches are generated depends on the consumption order constraints between the EPs, which comes from the order of the corresponding gates

in the circuit. Below, we formally define the consumption order over $\mathcal{E}$, problem formulation of generating $\mathcal{E}$ under given constraints, and then present our approaches.

Consumption Order $\mathcal{C}$ over EPs. Consider the set of EPs $\mathcal{E}$ that must be generated to execute the remote gates for a given qubit allocation. The consumption order $\mathcal{C}$ over the EPs in $\mathcal{E}$ is defined as follows. For $E_i, E_j \in \mathcal{E}$, we have $E_i \prec E_j$ if the gate g_i and g_j in C corresponding to the EPs E_i and E_j are such that they share an operand and g_i occurs before g_j in C. In addition, $\mathcal{C}$ is closed under transitive closure, i.e., if $E_i \prec E_j$ and $E_j \prec E_k$, then $E_i \prec E_k$. Note that the order $\mathcal{C}$ is a strict partial order; in particular, for no E_i, $E_i \prec E_i$.

Generation of EPs under Decoherence (GED) Problem. Given a set $\mathcal{E}$ of EPs with a consumption order $\mathcal{C}$, the EG problem is to partition the set $\mathcal{E}$ into subsets/batches $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_l$ (generated in that order) such that (i) each subset $\mathcal{E}_i$ has a total generation latency of at most τ, and (ii) the total circuit execution latency is minimized, assuming the batches are generated in the order $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_l$ and we use the GenerateEPs procedure to generate each batch.

No-Wait Property of GED Solutions. A GED solution $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_l$ satisfies the no-wait property if for all E_i, E_j pairs of EPs in $\mathcal{E}$, if $E_i \prec E_j$, then $E_i \in \mathcal{E}_i$ and $E_j \in \mathcal{E}_j$ such that $i \leq j$. For example, for Fig. 5, the GED solution $\{e_1, e_2, e_3, e_4\}, \{e_5, e_6, e_7\}, \{e_8\}$ satisfies the no-wait property. GED solutions with the above no-wait property ensure that the circuit execution time equals the sum of the generation latencies of the batches. Note that, in general, if the no-wait property is not satisfied, then the total circuit execution latency can be more than the sum of the generation latencies of the batches due to some already-generated EPs *waiting* for other EPs to be generated. Our designed approaches return solutions with the no-wait property.

In the following subsections, we develop dynamic programming (DP) and greedy approaches for the GED problem. The motivation behind our approaches is: (i) If the consumption order $\mathcal{C}$ is total, then an optimal approach based on DP can be designed. (ii) If no consumption order exists over EPs, then a greedy approach with performance guarantees under certain reasonable assumptions can be designed. Both approaches can be generalized for general partial consumption orders.

A. Dynamic Programming (DP) for Generating EPs

Here, we provide a dynamic programming (DP) algorithm for the GED problem; the DP algorithm is optimal for the special case when the consumption order $\mathcal{C}$ is total. We start with presenting the DP algorithm for the special case when the consumption order $\mathcal{C}$ is total; we then generalize the algorithm for the general $\mathcal{C}$ later.

Optimal DP Algorithm when $\mathcal{C}$ is Total. Let the set $\mathcal{E}$ of required EPs be $\{E_1, E_2, \ldots, E_m\}$ and without loss of generality, let the consumption order $\mathcal{C}$ be $E_1 \prec E_2 \prec E_3 \cdots \prec E_m$ and thus, be a total order. First, we observe (see Lemma 1 below) that, if the consumption order $\mathcal{C}$ is total, then there is an optimal GED solution that partitions $\mathcal{E}$ into "contiguous" batches—i.e., each batch is of the type $\{E_i, E_{i+1}, E_{i+2} \ldots E_{j-1}, E_j\}$ for $1 \leq i \leq j \leq m$.

Lemma 1: Consider a GED problem for a set of EPS $\mathcal{E} = \{E_1, E_2, \ldots, E_m\}$ and a total consumption order $\mathcal{C}$ as $E_1 \prec E_2 \prec E_3 \cdots \prec E_m$. There exists an optimal GED solution where each batch is a contiguous set of EPs. ■

Based on the above Lemma, finding the optimal GED solution for $\mathcal{E}$ with a total order $\mathcal{C}$ is tantamount to partitioning $\mathcal{E}$ into contiguous batches such that (i) each batch has a generation latency of less than the decoherence threshold τ, and (ii) the sum of generation latencies of the batches is minimized. Such an optimal partition of $\mathcal{E}$ can be found using a dynamic programming approach as follows. In particular, let us define S_j be the optimal partitioning of the subset $\{E_1, E_2, \ldots, E_j\}$ of EPs into contiguous batches such that (i) each batch has a generation latency of less than τ, and (ii) the sum of the latencies of the batches is minimum. Let $S[j]$ be the sum of generation latencies of the batches in S_j. Also, let L_{ij} be the total generation latency of the set of EPs $\{E_i, E_{i+1}, E_{i+2} \ldots E_{j-1}, E_j\}$ $(1 \leq i \leq j \leq m)$ when generated independently using the procedure GenerateEPs. Then, we have:

$$\begin{aligned} S[1] &= L_{11} \\ S[j] &= min_{i|i<j \text{ and } L_{ij} \leq \tau} \; S[i] + L_{ij} \end{aligned}$$

Using the above recursive equation, we can compute $S[j]$ (and the corresponding partitioning S_j) for all j, which yields the optimal solution S_n of the GED problem for $\mathcal{E}$.

Theorem 2: The above DP algorithm yields an optimal GED solution when $\mathcal{C}$ is total. ■

DP Algorithm for General $\mathcal{C}$. To generalize the above DP algorithm for a general consumption order $\mathcal{C}$, we first create a "topological sorting" of the EPs using the "edges" in $\mathcal{C}$ and then run the DP algorithm over the sorted/sequenced EPs. More formally, given a set of EPs $\mathcal{E}$ and a consumption order $\mathcal{C}$ over them, we first order and rename the EPs in $\mathcal{E}$ as $\langle E_1, E_2, \ldots, E_m \rangle$ such that for all E_i, E_j, if $E_i \prec E_j$ then $i < j$. Note that such an ordering is always possible since the consumption order $\mathcal{C}$ is a *strict* partial order (i.e., the $\mathcal{C}$ relation is acyclic). There are many such orders possible, and we pick any such order. After ordering the EPs as above, we run the above DP algorithm over the ordered list $\langle E_1, E_2, \ldots, E_m \rangle$ to determine the GED solution. It is easy to see that the DP solution satisfies the no-wait property, even for the general $\mathcal{C}$.

B. Greedy Approach for Generating EPs

We now design a greedy algorithm for the GED problem. We start by considering the special case when $\mathcal{C}$ is null.

Greedy′: Greedy Algorithm when $\mathcal{C}$ is Null. For the special case when $\mathcal{C}$ is null, at a high-level, the greedy algorithm iteratively picks a batch of EPs (from the remaining EPs in $\mathcal{E}$) with the lowest average latency. It can be shown that such a greedy algorithm would deliver an $O(\log(n))$-approximate solution, where n is the number of gates. However, determining the batch that has the minimum average latency is NP-hard; thus, within each iteration, we use an (inner-level) greedy approach

to determine the batch with *near-minimum* average latency. We now present our greedy approach (called `Greedy'`) in more detail, and and present performance guarantee results. Let $\mathcal{E} = \{E_1, \ldots, E_m\}$ be the set of EPs that need to be generated as an input to the `GED` problem, and the consumption order $\mathcal{C}$ be null. Then, our proposed `Greedy'` algorithm is as follows.

`Greedy':`

1) Let $S = \mathcal{E}$, the set of remaining EPs.
2) $i = 1$. /* The below generates the batch $\mathcal{E}_i$ */
3) Initialize $\mathcal{E}_i = E$, where E is the EP in S of lowest latency.
4) Initialize $S' = S$.
5) For $k = 1$ to $|S|$: /* Iteratively remove an EP E from S' that reduces the latency of remaining S' as much as possible. Update $\mathcal{E}_i$ to S', if needed.*/
   a) Remove the EP E from S' that reduces its latency the most. That is, pick an E in $\bar{S}$ that minimizes Latency$(S' - \{E\})$.
   b) $S' = S' - \{E\}$.
   c) If S' has latency less than τ and has lower average latency than $\mathcal{E}_i$, then $\mathcal{E}_i = S'$.
6) $S = S - \mathcal{E}_i$.
7) If $S = \{\}$ RETURN $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_i$, ELSE Increment i and Go to Step #3.

We use `GenerateEPs` procedure to determine the total latency of any set of EPs in the above `Greedy'` algorithm. Note that since $\mathcal{C}$ is null, the `Greedy'` solution trivially satisfies the no-wait property; hence, the circuit execution time equals the sum of the latencies of the batches in the solution.

Performance Guarantee Results. We make the following observations (we omit the proofs here). (i) If, in each iteration above, $\mathcal{E}_i$ is indeed the subset of EPs with the lowest average latency, then the `Greedy'` approach delivers an $O(\log(n))$-approximate solution. (ii) If the `GenerateEPs` procedure corresponds to a submodular function, then, in each iteration, the selected $\mathcal{E}_i$ is such that, for a given $|\mathcal{E}_i|$, $(S - \mathcal{E}_i)$'s latency is at least 63% of the maximum possible. In other words, the $\mathcal{E}_i$ is derived by removing a near-optimal set of EPs from S.

`Greedy`: Greedy Approach for a General $\mathcal{C}$. The `Greedy'` algorithm can be generalized to a general consumption order $\mathcal{C}$. The key goal of our generalized algorithm (called `Greedy`) is to preserve the no-wait property of the solution so that the circuit execution time doesn't include additional wait times. In particular, we modify the above greedy approach as follows. In Line 5 of the `Greedy'` 's pseudo-code above, which selects near-optimal batch $\mathcal{E}_i$ by iteratively removing an appropriate EP E, we instead remove E with all its "descendants" (i.e., all EPs E_j's in S s.t. $E \prec E_j$); more formally, in Line 5a, we pick an E that minimizes Latency$(S' - E')/|S' - E'|$ where E' is E and all its descendants. Removal of descendants ensures that the selected batch $\mathcal{E}_i$ doesn't include any EP that "depends" on another EP not yet included in a batch—and thus, ensures that the eventual solution satisfies the no-wait property. For example, consider the first iteration of `Greedy` over the EPs in Fig. 5. When considering removal of e_4, we must also remove $\{e_6, e_7, e_8\}$, leaving us with $\{e_1, e_2, e_3, e_5\}$. If e_4 is indeed chosen for removal, then we continue the algorithm over $\{e_1, e_2, e_3, e_5\}$, while keeping $\{e_1, e_2, e_3, e_5\}$ as a potential choice for $\mathcal{E}_1$.

VII. Cat-Entanglements to Execute Remote Gates

We have considered telegates to execute remote gates until now. We now consider using cat-entanglements instead, which can result in lower circuit execution time since a single cat-entanglement can enable the execution of several remote gates.

Given a qubit allocation, several possible sets of CEs may be sufficient to execute the required remote gates; thus, in this section, we discuss the problem of selecting a near-optimal set of CEs sufficient to execute the remote gates. The selected CEs yield the EPs to be generated; these EPs are then batched and ordered using approaches discussed in the previous section. We start with defining cat-entanglements.

Cat-Entanglements (CEs). A cat-entanglement (CE), requiring one EP, creates a read-only linked copy of a qubit q at another network node. Such a linked copy can be used as a control-qubit operand to execute gates until there is a unary gate on the qubit q, at which point the linked copy must be "destroyed" using the disentanglement operation (which doesn't require an additional EP). Thus, though telegates and CEs both require a single EP, a telegate helps execute one remote gate, while a CE may help execute several remote gates. Thus, using CEs can drastically reduce the set/number of EPs required and, thus, the circuit execution time.

Notation. A *cat-entanglement* is a triplet (q_i, P_k, t) signifying creation of a linked copy of qubit q_i on node P_k at circuit's time instant t. Disentanglement operations are implicit, i.e., done immediately before any unary operation on q_i. For simplicity, we assume that the circuit contains only unary and *CZ* gates, as the symmetry of *CZ* gates facilitates a simpler description; CNOT gates can be handled similarly (§IX).

A. Selecting CEs (and EPs) to Execute Remote Gates

The CE-selection problem is a generalization of the classical set-cover problem, as discussed below. We start with formalizing how CEs enable gate execution.

Execution (Coverage) of a Remote Gate by CEs. To execute a remote gate (q_i, q_j, t), we can either: (i) Create a linked copy of q_i (q_j) in the computer where q_j (q_i) is located, or (ii) Create linked copies of q_i and q_j in a third computer where the gate operation can be performed. Thus, a remote gate execution can be enabled by either a single CE or a pair of CEs. We say a single CE or a pair of CEs *cover* a gate if they enable the execution of the gate. We also define the cost $c(M)$ of a CE M as the generation latency of the EP needed for M.

CE Selection (`CESelection`) Problem. Given a qubit-allocation function η, the `CESelection` problem is to select a set of CEs $\mathcal{M} = \{M_1, M_2, \ldots M_r\}$ such that

- Every remote gate arising from the qubit-allocation function η is covered by CE(s) in $\mathcal{M}$.

- The total latency of the EPs corresponding to the CEs is minimized, i.e., $\sum_{i=1}^{r} c(M_i)$ is minimized.[2]

The above CESelection problem has been addressed in [13] for the special case when the objective is to minimize the *number* of CEs selected; here, we generalize their approach for the above objective of minimizing the *total cost* of CEs.

$O(\log(n))$-Approximation Greedy-CE Algorithm. The above CESelection problem can be looked upon as a weighted set-cover problem, wherein we need to select a minimum-weighted (minimum-cost, here) collection of given sets (CEs, here) to cover all the elements (remote gates, here). However, unlike the set-cover problem, in our case, an element (remote gate) may be covered by a *combination* of two sets (CEs); this renders the simple greedy algorithm (which picks the "best" set in each iteration) without any performance guarantee. However, a more sophisticated Greedy-CE algorithm that, in each iteration, ***picks a set of CEs*** that covers the most number of non-yet-covered remote gates per unit cost of the CEs picked—can be shown to deliver a $O(\log(n))$-approximate solution (see Theorem 3). However, we need to solve the non-trivial problem of picking such an optimal *set* of CEs; fortunately, this can be formulated[3] and solved using a generalized *densest subgraph* problem, as discussed below.

Densest Subgraph with Vertex-Costs (DSVC) Problem. Let $G = (V, E)$ be a graph where each vertex $v \in V$ has a weight $w(v)$ and a cost $c(v)$ associated, and each edge e has a weight $w(e)$ associated. The DSVC problem is find an induced subgraph H in G with maximum value of $(\sum_{e \in E(H)} w(e) + \sum_{v \in V(H)} w(v)) / \sum_{v \in V(H)} c(v)$. For the special case of the DSVC problem, when each vertex has a unit cost, [13] proposed two algorithms: an optimal linear programming and a 2-approximate greedy approach. Both algorithms and their performance guarantees can be generalized to include arbitrary vertex costs; due to limited space, we discuss only the 2-approximate greedy approach here. The DSVC-Greedy algorithm to solve the above DSVC problem is to iteratively remove a vertex v that has the lowest value of $\frac{w(v) + \sum_{e \text{ incident on } v} w(e)}{c(v)}$. We keep track of the remaining subgraphs over the iterations and pick the best among them (i.e., the subgraph H with the maximum value of $(\sum_{e \in E(H)} w(e) + \sum_{v \in V(H)} w(v)) / \sum_{v \in V(H)} c(v)$). We can show that DSVC-Greedy delivers a 2-approximate solution.

Lemma 2: The above DSVC-Greedy algorithm returns a 2-approximate solution for the DSVC problem. ■

Theorem 3: The above Greedy-CE algorithm for the CESelection problem delivers a $\mathcal{O}(\log n)$-approximate solution, where n is the total number of circuit gates. ■

[2] In general, as discussed before, the total (concurrent) generation latency of a set of EPs may *not* be the sum of the independent latencies. Still, we assume so for simplicity in formulating the CESelection problem.

[3] Consider a graph over CEs as vertices, where the weight on each vertex/CE v is the number of remote gates covered by v by itself and the weight on edge (u, v) is the number of gates covered by the pair of CEs $\{u, v\}$. However, a gate may be covered in many ways—which can lead to double counting; this can be resolved by partitioning the graph appropriately [13].

VIII. Other Generalizations

Non-Zero Swap and Gate Latencies. To incorporate non-zero swap and gate latencies, we make two changes. (i) When generating the EPs in batches, the gates and swaps are done in parallel with the generation of EPs of the *following* batch. Thus, gates and swaps do not add to the total execution time—as long as their latencies are less than the EP-batch latencies (which is expected to be largely true). (ii) We only consider batches whose total latency (including swap and gate operations) is less than the decoherence threshold τ.

Fidelity Constraints. The fidelity of EPs may degrade during the generation process due to the quantum operations (decoherence can also affect fidelity, but we have handled it separately). In our schemes, we use the purification technique to overcome fidelity degradation by requiring and creating multiple copies of EPs [10] (instead of just one) to execute each remote gate; we required the number of copies to be proportional to the entanglement path length l used to generate the EP (as fidelity degradation is somewhat proportional to l).

IX. Evaluation

In this section, we evaluate our algorithms over NetSquid [6], a QN simulator, on random and benchmark circuits.

Algorithms Compared. We compare our techniques with the algorithm proposed in [8] (referred as Disjoint-Paths), as it is the only work that considers the DQC-QR problem over general networks. We use our qubit-allocation scheme (§V) as a precursor to all the algorithms, including Disjoint-Paths (as [8] assumes a given qubit allocation). Our DQC-QR algorithms are named based on the execution schemes as follows: (i) Greedy-TG using telegates (§VI-B), (ii) DP-TG using telegates (§VI-A), (iii) Greedy-CE that selects near-optimal number of CEs (§VII) and then uses the Greedy approach over the selected CEs, and (iv) DP-CE, similarly. In our evaluations, we relaxed the requirement of fixed qubit allocation by not performing the reverse swaps; we observed this had minimal performance impact.

Generating Random Circuits and Benchmark Circuits. We evaluate the techniques on random quantum circuits using the following parameters: number of qubits (default=50), number of gates per qubit (default=50), and fraction of binary gates (default=0.5). Given the parameter values, we generate the random circuit one gate at a time. Gate operands are chosen randomly. We also evaluate on benchmark circuits corresponding to Quantum Fourier Transform (QFT), Quantum Phase Estimation (QPE), and GHZ state generation (GHZ), of various sizes obtained from the Munich Quantum Toolkit [20].

CNOT and CZ Circuits. Binary gates in the random circuits can be either all CNOT (referred to as CNOT circuits) or all CZ gates (referred to as CZ circuits). The benchmark circuits have only CNOT and unary gates. The Greedy-CE and DP-CE schemes can be applied directly to CZ circuits; for CNOT circuits, we first convert each $CNOT$ gate to a CZ gate and two unary gates before applying the CE schemes.

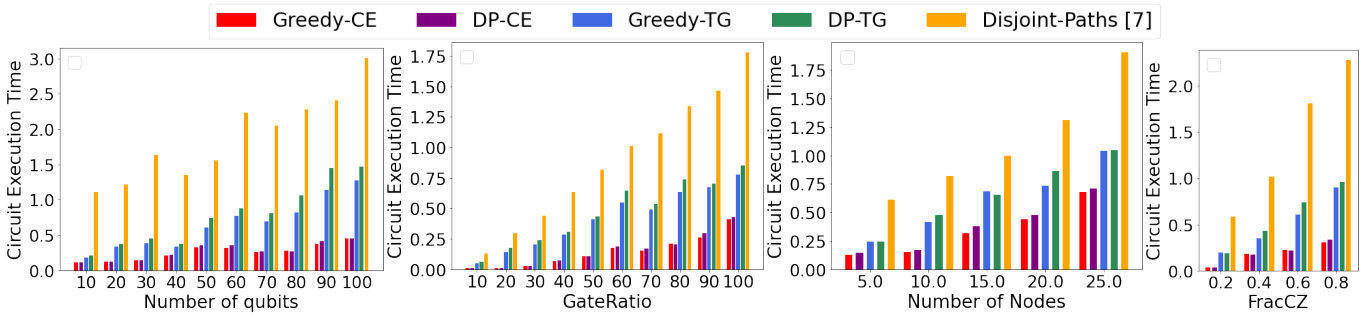

Fig. 6: Total execution time taken by various algorithms for varying parameters in *CZ* circuits.

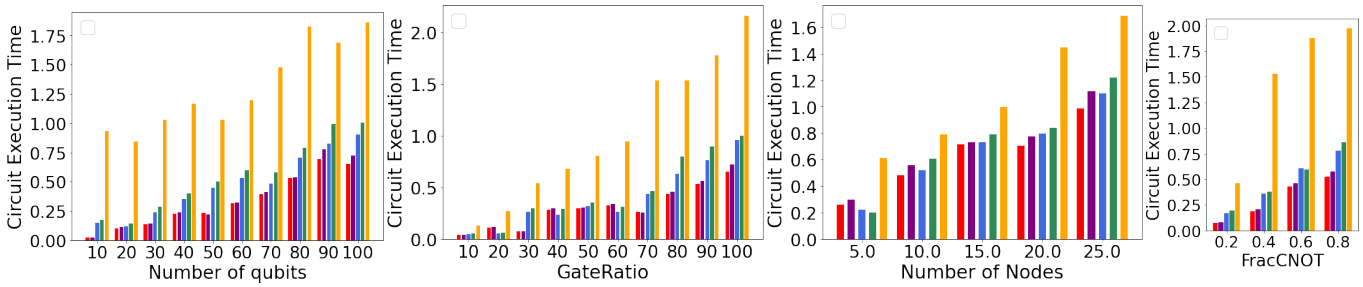

Fig. 7: Total execution time taken by various algorithms for varying parameters in $CNOT$ circuits.

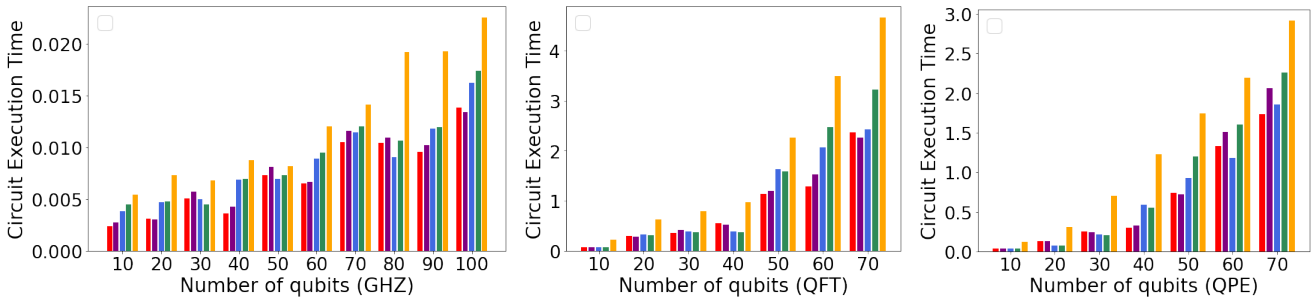

Fig. 8: Total execution time taken by various algorithms for ($CNOT$) benchmark circuits.

Generating Random Networks. We use a quantum network spread over an area of $100km \times 100km$. We use the Waxman model [26], which has been used to create Internet topologies. We vary the number of network nodes from 5 to 25, with 10 as the default. The total number of data memories in the network is equal to the number of circuit qubits; these data memories are randomly distributed among the network nodes.

Network Parameters. We use network parameter values similar to the ones used in [15]. 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.

Evaluation Results. We evaluate the algorithms over the circuits and networks as described above. We consider $CNOT$ and CZ gate-based circuits and vary one parameter at a time while keeping the other parameters fixed to their default values mentioned above. See Figs. 6-8. We observe the following:

- Generally, the performance of the algorithms is in the following order (best to worst): `Greedy-CE`, `DP-CE`, `Greedy-TG`, `DP-TG`, and `Disjoint-Paths`, with our `Greedy-CE` outperforming `Disjoint-Paths` [8] by up to 95% in some cases (see below).
- As expected, using CEs significantly reduces execution time, especially in *CZ* circuits. In the CNOT circuits, the CE schemes perform worse sometimes than the telegate schemes.

We believe there are **two key reasons for sometimes drastic under-performance** of `Disjoint-Paths` [8]: (i) Their layering strategy is very conservative in exploiting concurrency among required EPs; (ii) For an actual generation of EPs, they select one path for each EP's generation—focusing on maximizing the number of disjoint paths in a batch rather than generation latencies. In contrast, our schemes use sophisticated approaches to divide the set of required EPs into batches and then use the optimal LP scheme to generate each batch together through an optimal network flow of entanglements.

Runtime Overhead. Our schemes, being polynomial-time, run in order of a few seconds for large circuits; this is a tolerable overhead, especially since these distribution strategies need to be run only one time (for a given quantum algorithm).

X. Conclusion

In this paper, we have addressed the overarching problem of distribution of quantum circuits in quantum networks to minimize execution time, under decoherence and network resource constraints. Our future work focuses on developing provably near-optimal algorithms for the sub-problems addressed here, and in particular, to develop more sophisticated techniques that allow for dynamic qubit allocations.

References

[1] Pablo Andres-Martinez and Chris Heunen. Automated distribution of quantum circuits via hypergraph partitioning. *Physical Review A*, 100(3):032308, 2019.

[2] Esther M Arkin, Refael Hassin, and Maxim Sviridenko. Approximating the maximum quadratic assignment problem. *Information Processing Letters*, 77(1):13–16, 2001.

[3] Charles H Bennett, Gilles Brassard, Claude Crépeau, Richard Jozsa, Asher Peres, and William K Wootters. Teleporting an unknown quantum state via dual classical and einstein-podolsky-rosen channels. *Physical review letters*, 70(13):1895, 1993.

[4] H-J Briegel, Wolfgang Dür, Juan I Cirac, and Peter Zoller. Quantum repeaters: the role of imperfect local operations in quantum communication. *Physical Review Letters*, 81(26):5932, 1998.

[5] Andrew M. Childs, Eddie Schoute, and Cem M. Unsal. Circuit Transformations for Quantum Architectures. In *14th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2019)*, volume 135 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 3:1–3:24, 2019.

[6] Tim Coopmans et al. Netsquid, a discrete-event simulation platform for quantum networks. *Commun. Physics*, 2020.

[7] Alexander Cowtan, Silas Dilkes, Ross Duncan, Alexandre Krajenbrink, Will Simmons, and Seyon Sivarajah. On the qubit routing problem. In *Theory of Quantum Computation, Communication, and Cryptography*, 2019.

[8] Daniele Cuomo, Marcello Caleffi, Kevin Krsulich, Filippo Tramonto, Gabriele Agliardi, Enrico Prati, and Angela Sara Cacciapuoti. Optimized compiler for distributed quantum computing. *ACM Transactions on Quantum Computing*, 4(2), feb 2023.

[9] Omid Daei, Keivan Navi, and Mariam Zomorodi-Moghadam. Optimized quantum circuit partitioning. *International Journal of Theoretical Physics*, 59(12):3804–3820, 2020.

[10] W Dür and H J Briegel. Entanglement purification and quantum error correction. *Reports on Progress in Physics*, 70(8), Jul 2007.

[11] Jens Eisert, Kurt Jacobs, Polykarpos Papadopoulos, and Martin B Plenio. Optimal local implementation of nonlocal quantum gates. *Physical Review A*, 62(5):052317, 2000.

[12] Davide Ferrari, Angela Sara Cacciapuoti, Michele Amoretti, and Marcello Caleffi. Compiler design for distributed quantum computing. *IEEE Transactions on Quantum Engineering*, 2:1–20, 2021.

[13] Ranjani G Sundaram, Himanshu Gupta, and CR Ramakrishnan. Efficient distribution of quantum circuits. In *35th International Symposium on Distributed Computing (DISC 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[14] Mohammad Ghaderibaneh, Caitao Zhan, Himanshu Gupta, and CR Ramakrishnan. Efficient quantum network communication using optimized entanglement swapping trees. *IEEE Transactions on Quantum Engineering*, 3:1–20, 2022.

[15] Mohammad Ghaderibaneh, Caitao Zhan, Himanshu Gupta, and CR Ramakrishnan. Efficient quantum network communication using optimized entanglement swapping trees. *IEEE Transactions on Quantum Engineering*, 3:1–20, 2022.

[16] Gushu Li, Yufei Ding, and Yuan Xie. Tackling the qubit mapping problem for nisq-era quantum devices. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1001–1014, 2019.

[17] Yingling Mao, Yu Liu, and Yuanyuan Yang. Probability-aware qubit-to-processor mapping in distributed quantum computing. In *Proceedings of the 1st Workshop on Quantum Networks and Distributed Quantum Computing*, pages 51–56, 2023.

[18] Yingling Mao, Yu Liu, and Yuanyuan Yang. Qubit allocation for distributed quantum computing. In *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*, pages 1–10. IEEE, 2023.

[19] Eesa Nikahd, Naser Mohammadzadeh, Mehdi Sedighi, and Morteza Saheb Zamani. Automated window-based partitioning of quantum circuits. *Physica Scripta*, 96(3):035102, 2021.

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

[21] Sartaj Sahni and Teofilo Gonzalez. P-complete approximation problems. *Journal of the ACM (JACM)*, 23(3):555–565, 1976.

[22] Marcos Yukio Siraichi, Vinícius Fernandes dos Santos, Caroline Collange, and Fernando Magno Quintão Pereira. Qubit allocation as a combination of subgraph isomorphism and token swapping. *Proceedings of the ACM on Programming Languages*, 3(OOPSLA):1–29, 2019.

[23] Seyon Sivarajah, Silas Dilkes, Alexander Cowtan, Will Simmons, Alec Edgington, and Ross Duncan. t-ket: a retargetable compiler for NISQ devices. *Quantum Sci. and Tech.*, 2020.

[24] Ranjani G Sundaram and Himanshu Gupta. Distributing quantum circuits using teleportations. In *2023 IEEE International Conference on Quantum Software (QSW)*, pages 186–192. IEEE, 2023.

[25] Ranjani G Sundaram, Himanshu Gupta, and CR Ramakrishnan. Distribution of quantum circuits over general quantum networks. In *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 415–425. IEEE, 2022.

[26] Bernard M Waxman. Routing of multipoint connections. *IEEE journal on selected areas in communications*, 6(9):1617–1622, 1988.

[27] Anocha Yimsiriwattana and Samuel J. Lomonaco. Generalized ghz states and distributed quantum computing. *arXiv: Quantum Physics*, 2004. URL: https://api.semanticscholar.org/CorpusID:14682517.

[28] Mariam Zomorodi-Moghadam, Mahboobeh Houshmand, and Monireh Houshmand. Optimizing teleportation cost in distributed quantum circuits. *International Journal of Theoretical Physics*, 57:848–861, 2018.

[29] Alwin Zulehner, Alexandru Paler, and Robert Wille. An efficient methodology for mapping quantum circuits to the ibm qx architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(7):1226–1236, 2018.