

Efficient Execution of Multiple Quantum Circuits over a Quantum Network

Yukun Yang*
Stony Brook University, NY

Ranjani G. Sundaram*
Stony Brook University, NY

Himanshu Gupta
Stony Brook University, NY

Abstract—As quantum computing continues to scale, the ability to execute quantum circuits across distributed quantum networks is becoming increasingly important. While prior work has largely focused on distributing a single circuit to optimize the number of entanglement pairs (EPs) used or the execution time, future applications will require the efficient scheduling and execution of multiple circuits on a shared quantum network. Therefore, we study the problem of efficiently distributing multiple quantum circuits across a shared quantum network under decoherence and network constraints and seek to minimize the execution time required to execute all circuits (makespan).

Solving the above problem involves jointly determining when and where each circuit should be executed, and how to schedule concurrent EP generation required to execute remote gates. We propose several algorithmic approaches for this multi-circuit distribution problem and provide theoretical performance guarantees for special cases. To assess the practical effectiveness of our methods, we conduct extensive simulations using the NetSquid quantum network simulator.

I. Introduction

Current quantum hardware remains constrained by limited qubit counts and high noise levels, posing significant challenges for executing large-scale quantum circuits. Distributed quantum computing tackles these limitations by leveraging interconnected quantum devices to execute complex computations. This approach promises scalability and enhanced performance but introduces new challenges related to scheduling and execution time. First, distributed quantum computations require entanglements (to execute remote gates), which can incur significant generation latency and, thus, lead to decoherence of qubits. Second, in a multi-user environment, various applications will simultaneously demand access to quantum hardware available on the network. Without optimized scheduling and allocation mechanisms, this can lead to significant contention, delays, and underutilization of resources.

Thus, in this work, we consider the problem of distributing *multiple* 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). In our approaches, we first determine *when* and *how* to allocate each circuit, then,

use efficient strategies to execute the gates using required entanglements with minimum latency under network resource and decoherence constraints.

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 works on this subject have focused on distributing a *single* circuit to minimize the number of maximally-entangled pairs (EPs) either by minimizing the number of cat-entanglements [1, 9] or the number of teleportations [14, 18, 22]. A couple of recent works [4, 8, 16, 19] have focused on the more significant objective of minimizing the total latency incurred in generating the EPs needed for executing the remote gates. They, too, only consider the distribution of a single circuit over a network.

However, practical realization of quantum computation at scale will inevitably involve managing and executing multiple quantum circuits concurrently across quantum networks. This is partly due to multiple applications requesting use of shared network resources. Furthermore, many complex computation tasks (e.g., quantum simulations, variational quantum algorithms) are inherently modular and require repeated execution of several quantum subroutines either in parallel or in a specific sequence. Thus, it is necessary to consider the problem of efficiently executing multiple quantum circuits over a shared quantum network.

Our Contributions. In this paper, we formulate and address the problem of executing multiple quantum circuits on a quantum network while minimizing the total execution time through coordinated scheduling and resource allocation. Our key contributions are:

- **Problem Formulation and Batching Framework:** We formalize the Multi-Circuit Scheduling and Execution Problem (MC-SE) problem and introduce a batched execution paradigm (Theorem 1) that limits solutions to sequential execution of circuit batches. This approach reduces resource contention while maintaining near-optimal performance guarantees (logarithmic overhead) under static allocations and fixed entanglement generation times.
- **Algorithmic Techniques:**
 - *Exact and Approximate Batching Algorithms:* We develop `OptimalDP`, an exponential-time dynamic programming approach for optimal batch partitioning, and `GreedySC`, a set-cover-inspired algorithm with exponential runtime.

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

*These authors contributed equally to this work.

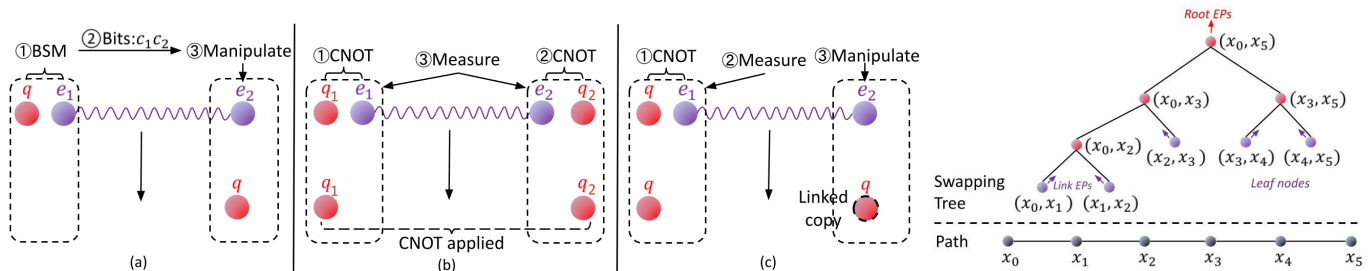


Fig. 1: Quantum communication. (a) Teleportation, (b) Telegate, (c) Cat-Entanglement. Dashed Fig. 2: Swapping tree to generate remote EPs. boxes are the network nodes; the initial (final) state of the qubits is at the top (bottom).

- *Specialized Polynomial Heuristics*: For practical scalability, we design polynomial-time heuristics leveraging merging strategies (Merging), incremental packing (Incremental), and polynomial heuristics for DP (OrderedDP) and set-cover (GreedySC-Heuristic, Densest-Batch-First).
- *Black Box Integration*: We demonstrate how single-circuit distribution algorithms from prior work can be repurposed as subroutines for batch latency estimation within our framework.
- **Theoretical Guarantees**: We prove that batched solutions incur at most logarithmic overhead compared to optimal unbounded scheduling under static qubit allocations (Section IV), and establish a 2-approximation guarantee for our greedy batching strategy on identical circuits.

Our work bridges the gap between single-circuit distribution algorithms and practical multi-circuit execution, enabling efficient utilization of quantum networks for concurrent workloads.

II. Background

Quantum Circuit Representation. We represent an *abstract quantum circuit* C over a set of qubits $\mathcal{Q} = \{q_1, q_2, \dots\}$ as a sequence of gates $\langle g_1, g_2, \dots \rangle$ 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, l) where q_i and q_j are the two operands and l is the index of the gate in the circuit, and unary gates as pairs (q_i, l) where q_i is the operand and l is the index.

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* [2] 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 [4]. 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 [6, 21]. 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. In our work, we use all three forms of communication to execute remote gates.

B. Generating EPs over Remote Nodes

Each of the above communication modes requires an a priori 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 [11, 17].

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 [3, 7] 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.

Discretized Time Steps. In order to schedule the placement and execution of circuits on the network, we discretize time to form time steps. The duration of each time step is assumed to be small enough to account for gate execution latency.

Qubit-Allocation Functions. Let the set of network nodes/QCs in the given quantum network be $\mathbb{P} = \{P_1, P_2, \dots, P_N\}$, where each QC P_i is equipped with a set $Q(P_i)$ of qubit memories for data storage. Additionally, let $C = \{c_1, c_2, \dots, c_n\}$ be the set of input circuits and $\{Q(c_1), Q(c_2), \dots, Q(c_n)\}$ be their sets of qubits, respectively. Consider a circuit c_i . Let s_i and e_i be the time steps at which c_i is first allocated and deallocated (after execution), respectively. Then, for every time step $t \in (s_i, e_i)$, we define a qubit allocation function $\mu_{(t,c)} : Q(c) \rightarrow \bigcup_i Q(P_i)$ as the function that maps the qubits of a circuit c to the network memories at time step t . We set $\mu_{(t,c)}$ to be null for all other time steps, i.e., $t \notin (s_i, e_i)$. Note that the qubit-allocation function may change during the circuit execution due to teleportations and/or swap operations.

A set of qubit allocations is considered **valid** if, at every time step t , at most one qubit occupies a network memory.

Remote and Local Gates. For a given circuit c and qubit allocation function $\mu_{(t,c)}$, a gate (q_i, q_j, l) in c is considered to be *remote at time step t* if q_i and q_j are assigned to different nodes by the qubit allocation function at time step t , i.e., $\mu_{(t,c)}(q_i) \in Q(P_i)$ and $\mu_{(t,c)}(q_j) \in Q(P_j)$ where $i \neq j$. Every gate that is *not* a remote gate is considered a *local gate*.

Multi-Circuit Scheduling and Execution Problem (MC-SE). Given a set of quantum circuits and a quantum network, determine the following entities such that the total execution time of all circuits is minimized and network/fidelity constraints are satisfied.

- **Start Times and Initial Qubit Allocations:** The start time s_j for each quantum circuit c_j and set of valid initial qubit allocations $\{\mu_{(s_j, c_j)}\}_{j \in [n]}$.
- **Execution Scheme:** Overall execution “scheme” of the circuits, which includes generation of necessary EPs (for all circuits) in an appropriate manner/order along with the execution of the circuits. Specifically, EPs need to be generated in an appropriate order—so that each generated EP can be consumed before it decoheres. 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 generated EPs to wait before being consumed. See [16] for more details.

The MC-SE problem can be shown to be NP-Hard by a reduction from the DQC problem [16].

Example 1. Fig. 3 shows an instance and solution of the MC-SE problem. Here, we present two possible methods to execute three circuits over the input network. In the first method, only two circuits are allocated initially, and when one of them finishes execution, we allocate the third circuit in the node vacated by the preceding circuit. In the second method, we allocate all three circuits at the same time. We observe that the first method takes two time steps to execute the three circuits, while the second method takes three time steps. This example motivates why proper scheduling of circuits is necessary for the MC-SE problem. In general, it may be better to defer allocation of a circuit even if there is sufficient remaining capacity in the network.

A. Related Work

The distribution of quantum circuits (DQC) problem has been studied before in various settings and objectives. Prior work can be categorized into two broad categories: works that minimize the *number* of EPs required and works that minimize the *circuit execution time*. We discuss these below.

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-QR problem, the DQC problem ignores the coupling graphs within each computer [1, 5, 9, 14], and thus, the swap operations needed. The optimization objective used in almost all of

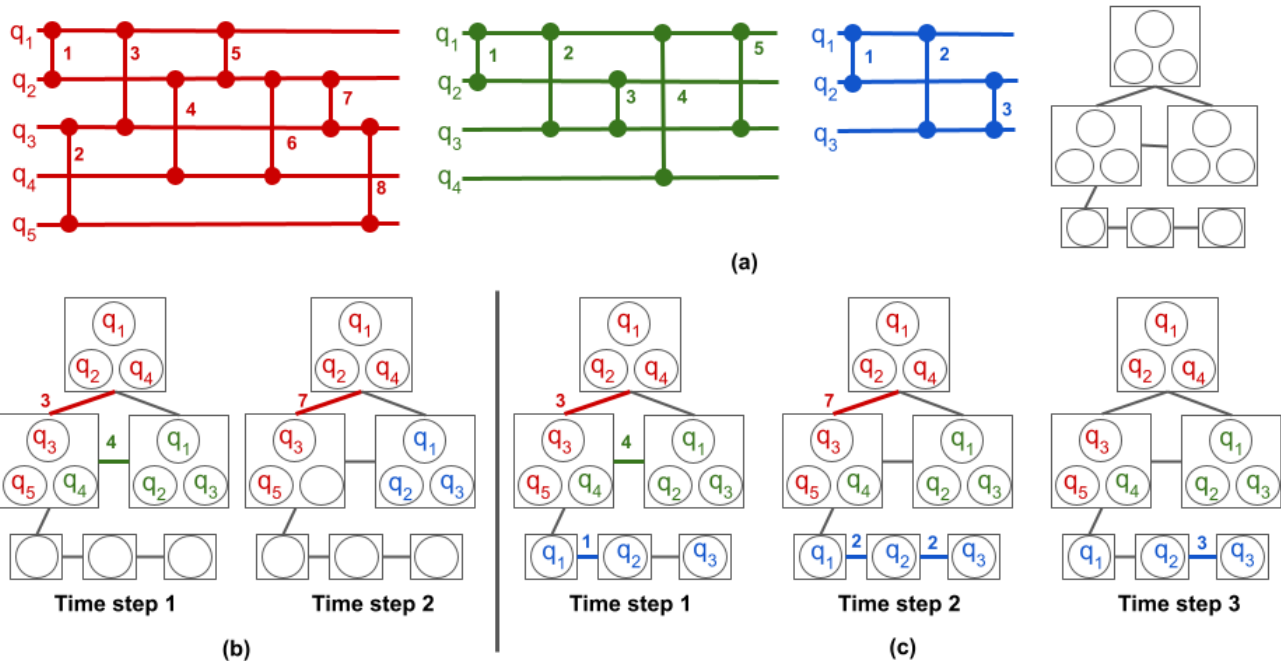


Fig. 3: MC-SE Problem Example 1. (a) Three quantum circuits and a quantum network. Here, the boxes represent network nodes and edges connecting boxes represent network links. We assume that executing local gates in this network incurs no latency. (b) and (c) represent two possible ways of scheduling and executing the three circuits. In both schemes, circuit qubits (colored) are placed in network memories (circles). The colored/labelled edges represent remote gates that are performed across network links. Note that each remote gate incurs latency equal to one time step, but two remote gates may be executed concurrently if they do not share network links. In (b), only the red and green circuits are allocated initially. Then, after time step 1, the green circuit is done executing (and thus, deallocated) and replaced by the blue circuit. Since all the gates in the blue circuit are made local by this allocation, the total execution time of the three circuits is two time steps. On the contrary, in (c), all three circuits are allocated initially. Here, executing the three circuits incurs three time steps.

these works has been to minimize the communication cost—defined as the *number* of EPs used; a couple of recent works [12, 13] 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 [9, 18, 20] 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; we also consider telegates to execute remote gates while the two-step DQC works consider entanglements [9, 20] and/or teleportations [18, 20].

Minimizing Circuit Execution Time. The works close to ours [4, 8, 16] address the DQC problem with minimizing circuit execution time, but with a static qubit allocation, slightly different settings, and certain limitations, as discussed below. In particular, [8] 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. [4] 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.

[16, 19] focus on minimizing the execution time of a *single* quantum circuit by concurrently generating EPs. Both works use a two-step algorithm- the first step determines a qubit

allocation by solving an instance of the maximum quadratic assignment problem. The second step creates an optimized sequence of batches of EPs. The EPs are then generated in this order, with gates corresponding to a batch being executed as soon as the EPs in the batch are generated.

Generalizing to Multiple Circuits. We generalize the DQC problem to incorporate multiple circuits, and seek to minimize the total execution time of all the circuits.

IV. Batched Execution of Circuits.

The QC-SE formulation in the previous section permits a circuit’s execution to start during the execution of other circuits. While this full generality is attractive in theory, it complicates reasoning about resource contention across the network. We therefore introduce a restricted class of solutions called *batched* solutions which offer near-optimal performance under certain assumptions.

Specifically, *batched solutions* are the MC-SE solutions wherein a circuit starts its execution *only* when no other circuits are currently executing in the quantum network. More formally, the start time s_i of any circuit c_i must satisfy $s_i < s_j$ or $s_i > e_j$ for all $j \neq i$; here, recall that s_j and e_j are the start and end times of a circuit c_j . Below, we show that batched solutions offer near-optimal (i.e., within an $O(\log n)$ factor, where n is the number of circuits) performance under the following assumptions:

- *Static qubit allocations*, i.e., the qubit allocation functions for any circuit are the same for all the time steps during its

execution; in effect, the static qubit allocation assumption disallows teleportation.

- *Partitioned qubit allocation.* There exists an optimal solution such that, for any two circuits c_i and c_j , the set of memories used by them is either disjoint or the same.
- *Fixed EP Latency,* i.e, the generation latency for an EP over a given pair of network nodes is a constant; in effect, the generation latency of an EP does not depend on the network state (specifically, other EPs being generated at that time).

Theorem 1: For an MC-SE instance with n input circuits with an optimal latency of T^* , there exists a batched solution with a latency at most $T^* \log n$, under the assumptions of fixed EP latency, and static and partitioned qubit allocations.

PROOF: We are given an instance of the MC-SE problem with circuits c_1, \dots, c_n , and a network \mathbb{P} . Let T^* be the optimal latency of this instance. We use $\text{Latency}(c)$ to represent the latency incurred by executing c in the given optimal solution. Also, we use $\text{Latency}(B)$ to denote the latency of executing all circuits in B .

We now construct a batched solution that utilizes the same “mapping” from circuit qubits to network memories, i.e., the same static qubit allocation function but perhaps over a different execution duration. Thus, since we are only shifting the start times of the circuits while retaining the qubit mapping/allocation, the latency of each circuit is the same in both algorithms, due to fixed EP latency assumption. Moreover, the latency of a batch B is the maximum of the latencies of the circuits in it, i.e., $\text{Latency}(B) = \max_{c \in B} \text{Latency}(c)$.

Algorithm for Constructing Batches. We construct a batched solution from the given optimal solution iteratively as follows.

- Pick the circuit with the highest latency that can fit in the current batch (retaining the qubit-memory mapping/allocation from the optimal solution).
- If no circuits can be added to the current batch, start the next batch.

Let $\mathcal{B} = \{B_1, \dots, B_k\}$ be the batches generated from the above algorithm.

Claim. For each batch B_i , $\text{Latency}(B_i) \leq \frac{T}{i}$.

PROOF: Suppose the claim is not true. Then, we have $\text{Latency}(B_i) > \frac{T}{i}$. Therefore, there is a circuit c_i in B_i such that $\text{Latency}(c_i) > \frac{T}{i}$.

Let S_i denote the set of memories used by the circuit c_i in the optimal algorithm. Then, for each $j < i$, there is a circuit c_j in B_j such that

- 1) $S_j \cap S_i$ is non-empty.
- 2) $\text{Latency}(c_j) > \frac{T}{i}$ (otherwise, c_i would have been selected in iteration j .¹)

Additionally, since \mathcal{I} is an instance that has the partitioned qubit allocation property, $S_i \cap S_{j_1} \neq \emptyset$ and $S_i \cap S_{j_2} \neq \emptyset$ implies $S_{j_1} \cap S_{j_2} \neq \emptyset$, for every $j_1, j_2 \in [i]$. This means that

¹Consider every circuit $c_j \in B_j$ with $S_j \cap S_i \neq \emptyset$. If $\text{Latency}(c_j) \leq \frac{T}{i}$ for every such c_j , c_i would have been chosen before any of them.

the optimal algorithm has at least i circuits that cannot be executed in parallel, and each incurs a latency greater than $\frac{T}{i}$. This is not possible as the optimal latency is T . Thus, the claim is true. \square

From the above claim, we get:

$$\begin{aligned} \text{Total latency of } \mathcal{B} &= \text{Latency}(B_1) + \dots + \text{Latency}(B_k) \\ &\leq T + \frac{T}{2} + \dots + \frac{T}{n} \leq T \log n. \end{aligned}$$

Above, we have used the fact that there can be almost n batches ($k \leq n$). \blacksquare

Restricting attention to batched solutions greatly simplifies the search space without majorly affecting the performance. While this guarantee does not hold when we relax the fixed-time EP, static allocation, and clustered instance assumptions, batching still provides efficient solutions in practice. Therefore, the remainder of the paper considers only batched solutions. We now define the `Batched MC-SE` problem.

Batched MC-SE Problem. Given a quantum network and a set of circuits, determine the following entities such that the total execution time of all circuits is minimized and network/fidelity constraints are satisfied:

- **Batches:** an ordered list of batches $\{B_i\}$
- **Start Times:** the start time t_i for each batch B_i
- **Qubit Allocations:** a valid qubit allocation for every circuit at the start time of its batch.
- **Execution Scheme:** Overall execution “scheme” of the circuits, which includes generation of necessary EPs (for all circuits) in an appropriate manner/order along with the execution of the circuits.

Note that while Theorem 1 only holds under certain assumptions, batched solutions can incorporate general EP generation time and dynamic allocations. In the upcoming sections, we present various algorithms that exploit this structure.

V. High-Level Approach

The previous section formalized our task as the `Batched MC-SE Problem`: choose how to group the input circuits into sequential batches such that the total execution time is minimized. Our high-level approach to tackling this problem is as follows:

- Use an existing algorithm from [16] as a black box to obtain qubit allocations and execution schemes for a single circuit.
- To evaluate a candidate batch that contains several circuits, we *stitch* those circuits into a single “mega-circuit” and pass it along with the input quantum network to the black box algorithm.
- Focus on the problem of “how” to bundle circuits into batches that incur the lowest execution time.

Black Box Algorithm. We treat the `DPoverSubCircuits` algorithm from [19] as an oracle that, given a single circuit and a quantum network, returns

- 1) An initial qubit allocation (provided the entire circuit can fit in the network)
- 2) An execution scheme for the circuit (including selection for teleportations, telegates, and cat-entanglements) along with its expected latency.

Since our algorithms are independent of the internal workings of the black box subroutine, their theoretical guarantees rely on the assumption of an optimal black box. However, this agnosticism also implies that replacing the current black box with a more efficient subroutine has the potential to yield enhanced performance.

We now describe our approaches in detail.

A Special Case: Identical Circuits. A simple case for the Batched MC-SE problem is that the circuits are all identical. We present the optimal DP approach and a 2-approximate greedy approach with proof, both of which are polynomial time.

Optimal DP. Suppose we have N identical circuits to execute. Let $\text{OPT}[k]$ denote the minimal latency for executing k identical circuits, and L_k denote the latency for executing k circuits in a single batch. The DP solution follows:

- 1) *Initialization.*
 - Initialize DP table with $\text{OPT}[0] = 0$.
- 2) *Bottom-Up Computation.*
 - For $k = 1$ to N :
 - Compute $\text{OPT}[k] = \min_{0 \leq i < k} (\text{OPT}[i] + L_{k-i})$.
 - Track splitting point i^* that achieves the minimum.
 - Reconstruct batches through backtracking.

Its easy to see that the above is optimal for the special case of identical circuits.

2-approximate Greedy.

- 1) *Parameter Calculation:*
 - Find optimal batch size: $k_1 = \text{argmin}_{1 \leq k \leq N} \frac{L_k}{k}$.
 - Decompose total circuits: $N = p \cdot k_1 + r$ where $0 \leq r < k_1$.
- 2) *Batch Construction:*
 - Create p full batches of size k_1 .
 - Create 1 residual batch of size r (if $r > 0$).

PROOF: Let $t_N = pL_{k_1} + L_r$ be the greedy solution of total latency by N identical circuits. Since $L_r \leq L_{k_1}$, we have $t_N \leq (p+1)L_{k_1}$. By the definition of k_1 , we have $\text{OPT}[k_1] = L_{k_1}$ and $\text{OPT}[p \cdot k_1] = p \cdot L_{k_1}$. So $t_N \leq \text{OPT}[p \cdot k_1] + \text{OPT}[k_1] \leq 2\text{OPT}[N]$. ■

In the upcoming Sections VI and VII, we provide approaches for the general Batched MC-SE problem (non-identical circuits).

VI. Exponential Time Approaches

In the following, we present two exponential time approaches based on DP and set cover to solve the Batched MC-SE problem.

OptimalDP. This algorithm guarantees minimal total latency by exhaustively evaluating all possible partitions of circuits

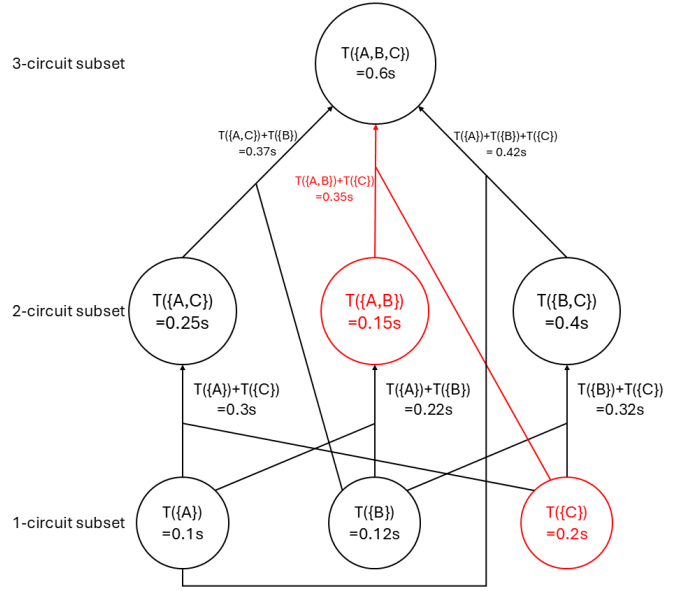


Fig. 4: **OptimalDP:** Recursive subset partitioning for circuits $\{A, B, C\}$. Circles represent possible batches and the connecting paths represent possible split. Red paths represent the optimal split.

into batches using DP. It recursively splits a subset of circuits S into two smaller subsets S_1, S_2 , computes their combined latency, and compares it with the batch latency of S . Although optimal, its exponential time complexity $O(F \cdot 2^n + 4^n)$ (F denotes the time to execute the blackbox once) makes it computationally prohibitive for large-scale circuits. It serves primarily as a theoretical baseline for small instances.

- 1) *Initialization.*
 - Precompute $\text{batch_time}(S) \leftarrow \text{blackbox}(S)^2$ for every subset $S \subseteq \{c_0, \dots, c_{n-1}\}$.
 - Initialize $\text{best_batch_collection}(S) \leftarrow [S]$ for all S .
- 2) *Subset Partitioning.*

For $k = 2$ to n :

 - For each k -circuit subset S :
 - Evaluate all bipartitions $\{S_1, S_2\}$ of S .
 - Update $\text{batch_time}(S) \leftarrow \min(\text{batch_time}(S), \text{batch_time}(S_1) + \text{batch_time}(S_2))$.
 - If improved, set $\text{best_batch_collection}(S) \leftarrow \text{best_batch_collection}(S_1) \cup \text{best_batch_collection}(S_2)$.
- 3) *Return.*
 - Return $\text{best_batch_collection}$ for the full circuit set with minimal latency.

Example. Suppose we have three circuits A, B, C and the batch latency (for simplicity we use the notation $T(S)$ to denote the latency of batch S) are $T(\{A\}) = 0.1s$, $T(\{B\}) = 0.12s$, $T(\{C\}) = 0.2s$, $T(\{A, B\}) = 0.15s$, $T(\{A, C\}) =$

²If S does not fit in the network (total qubit number exceed the total memory), blackbox will return infinity.

0.25s, $T(\{B, C\}) = 0.4\text{s}$, $T(\{A, B, C\}) = 0.6\text{s}$. We will use `OptimalDP` to obtain the minimal total latency of these circuits.

As shown in Fig. 4, `OptimalDP` evaluates all possible splits (black paths) but selects the minimal latency path (red path):

- Split $\{A, B, C\}$ into $\{C\}$ and $\{A, B\}$ (total $0.2 + 0.15 = 0.35\text{s}$).
- Keep $\{A, B\}$ as a single batch (better than splitting into $\{A\} + \{B\}$).

This gives an optimal total latency of 0.35s compared to 0.60s for a monolithic batch.

GreedySC. Modeled after the set cover problem, this algorithm greedily selects batches that minimize the latency per circuit. For each uncovered circuit, it identifies the subset S containing the circuit with the lowest $\text{blackbox}(S)/|S|$, iteratively covering all the circuits. Although faster than `OptimalDP`: $O(F \cdot 2^n + 2^n)$, its quality depends on circuit patterns and may overcommit to locally optimal batches.

- 1) *Initialization.*
 - Let $\text{Uncovered} \leftarrow \{c_0, \dots, c_{n-1}\}$.
 - Initialize $\text{Total_latency} \leftarrow 0$.
- 2) *Greedy Selection.*
 - For each $c \in \text{Uncovered}$:
 - Find subset S containing c that minimizes $\text{blackbox}(S)/|S|$.
 - Add S to batch_collection and update Total_latency .
 - Remove S from Uncovered .
- 3) *Return.*
 - Return batch_collection and Total_latency .

VII. Polynomial Time Approaches

Since exponential-time approaches are not scalable, we present heuristic versions of the algorithms in Section VI, along with additional heuristics from alternative perspectives, to solve the `Batched MC-SE` problem.

A. DP Heuristic

OrderedDP. This algorithm optimizes ordered circuit sequences using a DP approach inspired by matrix-chain multiplication. It computes the minimal latency for contiguous circuit ranges $[i, j]$ by evaluating splits at all possible points k , comparing the combined latency of $[i, k]$ and $[k+1, j]$ with the batch latency $[i, j]$. With $O(F \cdot n^3)$ complexity, it efficiently handles large sequences but is suboptimal when the order of circuits is random.

- 1) *Initialization.*
 - Define $\text{DP}[i][j]$ as the minimal latency for circuits c_i to c_j .
 - Initialize $\text{DP}[i][j] \leftarrow \text{blackbox}(\{c_i, c_{i+1}, \dots, c_j\})$.
- 2) *Bottom-Up Computation.*
For length $l = 2$ to n :
 - For start $i = 0$ to $n - l$:

- Let $j = i + l - 1$.
- For split $k = i$ to $j - 1$:
 - * If $\text{DP}[i][k] + \text{DP}[k+1][j] < \text{DP}[i][j]$, then update:
 $\text{DP}[i][j] \leftarrow \text{DP}[i][k] + \text{DP}[k+1][j]$

3) *Return.*

- Return $\text{DP}[0][n-1]$ and reconstruct batches from split history.

B. SetCover Heuristic

GreedySC-Heuristic. This method mimics human trial-and-error batching strategies. It incrementally constructs batches by expanding the lowest-latency circuit, adding candidates that reduce marginal latency. Starting with c_{\min} , it evaluates whether merging a new circuit c lowers total latency compared to separate execution ($\text{blackbox}(S \cup \{c\}) \leq \text{blackbox}(S) + \text{blackbox}(\{c\})$). With $O(F \cdot n^2)$ complexity, it balances local optimization and scalability but may miss globally synergistic batches.

- 1) *Batch Initialization.*
 - Start a new batch with the lowest-latency unassigned circuit c_{\min} .
- 2) *Circuit Addition.*
 - For each remaining unassigned c :
 - Add c to the current batch if $\text{blackbox}(S \cup \{c\}) \leq \text{blackbox}(S) + \text{blackbox}(\{c\})$.
 - Terminate the current batch if no such c exists.
- 3) *Repeat Until Complete.*
 - Mark all circuits in the current batch as assigned.
 - If there are unassigned circuits remaining, return to Step 1 to begin a new batch.
 - Otherwise, terminate and return all batches and total latency.

Densest-Batch-First. By simulating execution on a network augmented with high-cost dummy nodes, this heuristic implicitly identifies dense circuit clusters. It runs all circuits on the augmented network to find out the tightly coupled circuits which are assigned to the quantum memory on the original network for cost efficiency, and group them into a batch. Although explicit batching logic is avoided, its performance depends on the ability of the blackbox and the structure of the augmented network.

- 1) *Network Augmentation.*
 - Add a high-cost clique of unit-memory nodes to the original network N to form N' .
- 2) *Batch Extraction.*
 - Stitch all circuits into a super-circuit C' and run on N' .
 - Extract batches from sub-circuits assigned to N .
- 3) *Repeat.*
 - Recurse on remaining circuits until all circuits are batched.
- 4) *Return.*
 - Use this batch collection to get the total latency.

C. Other Heuristics

Incremental. This beam-search heuristic maintains diversity by retaining the top- K partial solutions at each step, avoiding premature commitment to suboptimal batches. For each circuit, it generates new solutions by adding this circuit to existing batches or creating a new one, pruning all but the K best candidates. With $O(F \cdot K \cdot n^2)$ complexity, it offers tunable trade-offs between optimality and speed, particularly effective for heterogeneous circuits.

1) Initialization.

- Let $DP[i]$ store up to K best batch collections for the first i circuits.
- Initialize $DP[0]$ with the trivial batch containing only the first circuit.

2) Iterative Expansion.

For $i = 1$ to $n - 1$:

- For each batch collection in $DP[i - 1]$:
 - Generate new solutions by adding c_i to existing batches or as a new batch.
- Prune to retain top- K solutions.

3) Return.

- Return the best batch collection in $DP[i - 1]$ with minimal total latency.

Merging. Starting with singleton batches, this algorithm iteratively merges pairs with the highest latency reduction, quantified by $\text{blackbox}(S_i) + \text{blackbox}(S_j) - \text{blackbox}(S_i \cup S_j)$. It greedily updates the batch set until no further gains exist, achieving $O(F \cdot n^3)$ complexity dominated by pairwise evaluations. Prone to local optima, it works well for moderate-sized problems.

1) Initialization.

- Treat each circuit as a singleton batch.

2) Merging Batches.

- While True:
 - Compute merging gain for all batch pairs: $\text{gain}(S_i, S_j) = \text{blackbox}(S_i) + \text{blackbox}(S_j) - \text{blackbox}(S_i \cup S_j)$.
 - Merge the pair with maximum positive gain.
 - Stop when all gain are negative.

3) Return.

- Return merged batches and total latency.

Example. Also consider the example used in `OptimalDP`. This time we will use `Merging` to obtain the minimal total latency of these circuits.

As shown in Fig. 5, `Merging` evaluates all possible batch pairs (black links) but selects the highest positive gain (red link):

- Merge $\{A\}, \{B\}$ into $\{A, B\}$.
- Stop merging and keep $\{A, B\}$ and $\{C\}$ as two batches (total $0.15 + 0.2 = 0.35$ s).

This gives an optimal total latency of 0.35s compared to 0.60s for a monolithic batch.

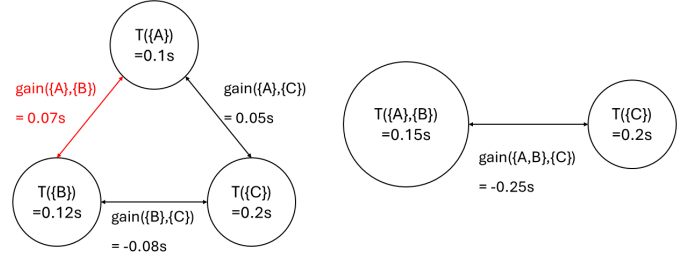


Fig. 5: `Merging`: Batch merging based on latency gain. Red link indicates the pairs with the highest positive gain selected for merging. Merging stops once all gains are negative.

VIII. Evaluation

In this section, we evaluate our algorithms on NetSquid, a QN simulator, over random small-scale and large-scale networks and circuits, as well as benchmark circuits, respectively.

Algorithms Compared. We compare our seven algorithms from Sections VI to VII using the blackbox subroutine proposed in [16], which by default estimates latency via telegates. For evaluation, we also switch the blackbox to use cat-entanglement. Although the blackbox can alternatively use teleportation to compute latency, we do not explore that option in this paper. Note that all algorithms are agnostic to the underlying blackbox model, as long as it implements the desired single-DQC algorithm.

Generating Random Circuits. For small-scale circuits instance, we generate random quantum circuits using the following parameters: number of circuits (default value: 10), number of qubits (default value: 20), number of gates per qubit (default value: 50) and fraction of binary gates (fixed value: 0.5). For big-scale circuits instance, we use the following parameters: number of circuits (default value: 500), number of qubits (fixed value: 100), number of gates per qubit (default value: 50) and fraction of binary gates (fixed value: 0.5). Given the values of the parameters, we generate the random circuit one gate at a time. Gate operands are chosen randomly. We also evaluate the benchmark circuits corresponding to Quantum Fourier Transform (QFT), Quantum Phase Estimation (QPE), Deutsch Jozsa, and GHZ state generation (GHZ), of various sizes obtained from the Munich Quantum Toolkit [15].

Generating Random Networks. The quantum network that we use is spread over an area of $100km \times 100km$. For a small-scale network instance, we use the following parameters: number of network nodes (default value: 5, for the execution of benchmark circuits, we fix it to 20), number of quantum memory per node (fixed value: 10). For a big-scale network instance, we use the following parameters: number of network nodes (fixed value: 50), number of quantum memory per node (fixed value: 20). The small- or big-scale networks and circuits will be paired respectively. During a batch execution, there may be some idle quantum memory.

Network Parameters. We use values of network parameters similar to those used in [11]. 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

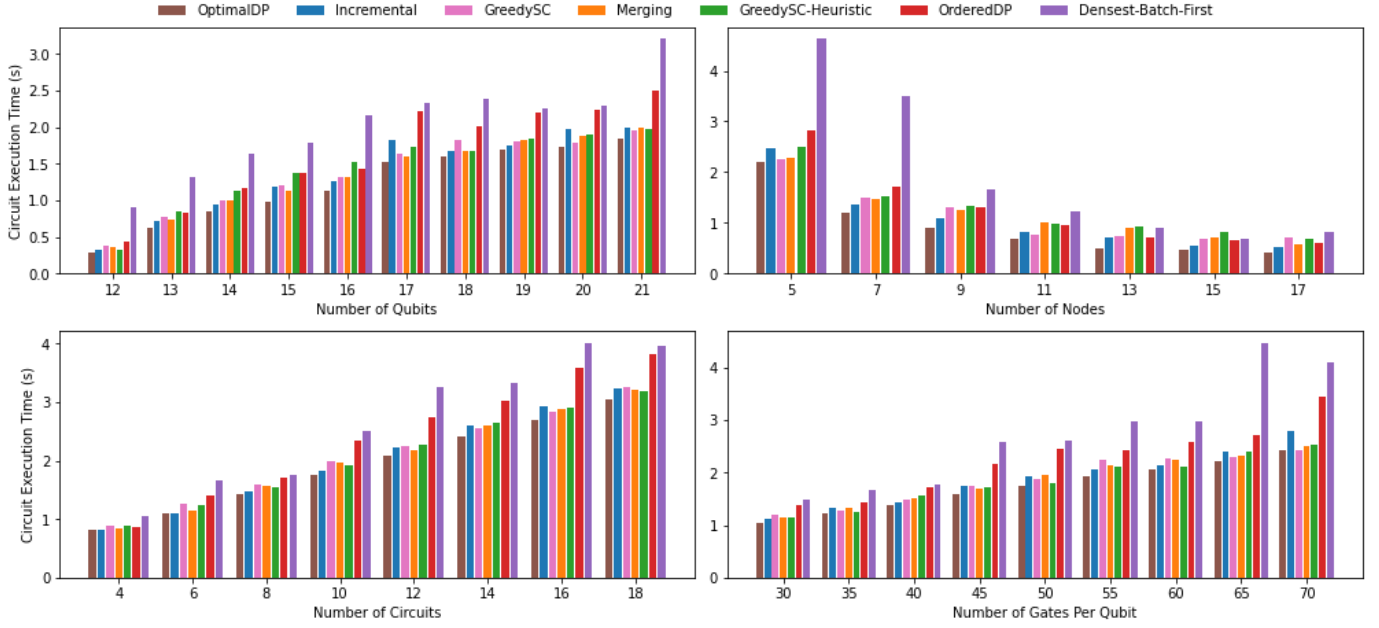


Fig. 6: Total execution time taken by various algorithms under different parameters for small-scale networks and circuits.

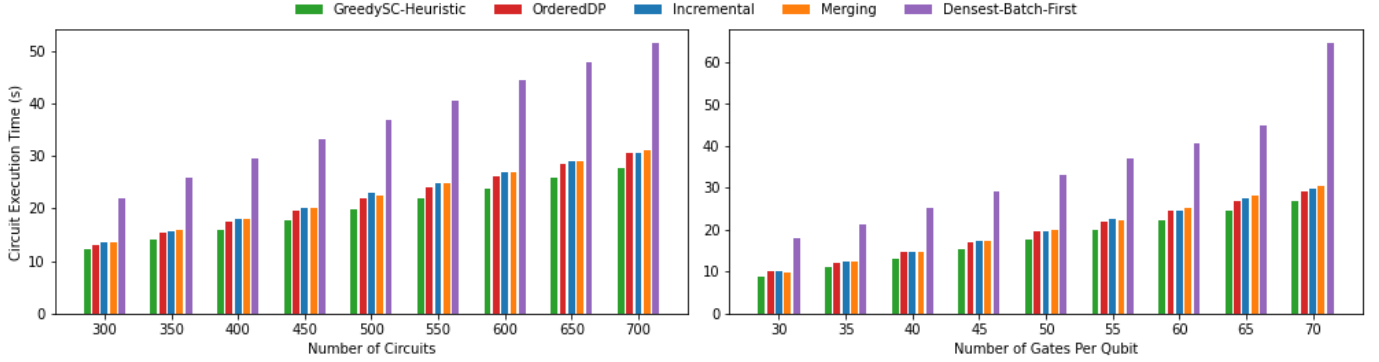


Fig. 7: Total execution time taken by various algorithms under different parameters for large-scale networks and circuits via telegate.

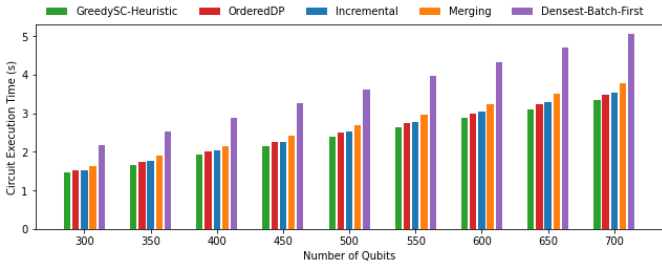


Fig. 8: Total execution time taken by various algorithms under different parameters for large-scale networks and circuits via cat-entanglement.

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 on the circuits and networks as described above, varying one parameter at a time while keeping the other parameters fixed to their default values. The exponential time algorithms are not applicable for

large-scale data. See Figs. 6-9. We observe the following:

1) For small-scale data (Figs. 6):

- OptimalDP is consistently the best. (Validating theoretical optimality, though exponential time).
- Densest-Batch-First and OrderedDP are consistently the worst, likely due to suboptimal partitioning strategies.
- The remaining four algorithms are similar: Incremental, Merging, GreedySC, GreedySC-Heuristic.

2) For large-scale data (Figs. 7- 8):

- Densest-Batch-First consistently does worst.
- GreedySC-Heuristic does best.
- The remaining three algorithms are similar: OrderedDP, Incremental, Merging.

3) For benchmark circuits (Figs. 9):

- OptimalDP is consistently the best.
- Densest-Batch-First and OrderedDP are consistently the worst.

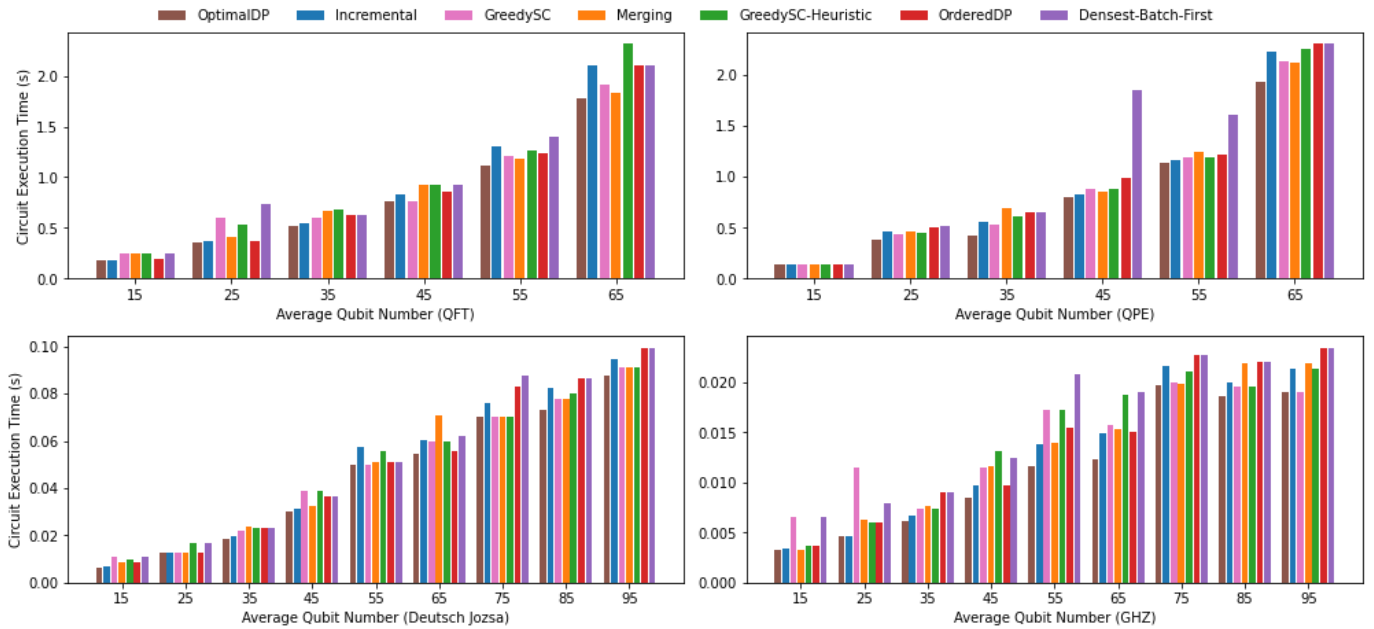


Fig. 9: Total execution time taken by various algorithms under different parameters for benchmark circuits.

- The remaining four algorithms are similar: Incremental, Merging, GreedySC, GreedySC-Heuristic.

Runtime Overhead. Our five heuristics, being polynomial-time, run on the order of a few seconds for small-scale data and a few minutes for large-scale data. This represents an acceptable performance for practical use.

IX. Conclusion

In conclusion, this work has demonstrated the near-optimality of a 'batched' approach for allocating and executing multiple quantum circuits over a network under specific settings. Furthermore, we provided several algorithms designed to tackle the problem of scheduling and executing collections of quantum circuits in a distributed manner, aiming to minimize the total execution time. In the future, we seek to extend our analysis to the more dynamic scenario of circuits arriving online, where proactive strategies such as the pre-distribution of entanglement pairs [10] could offer significant advantages in reducing latency. Additionally, we wish to explore and evaluate non-batched solutions to the MC-SE problem that might improve performance.

REFERENCES

- [1] Pablo Andres-Martinez and Chris Heunen. Automated distribution of quantum circuits via hypergraph partitioning. *Physical Review A*, 100(3):032308, 2019.
- [2] 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.
- [3] 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.
- [4] 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.
- [5] Omid Daei, Keivan Navi, and Mariam Zomorodi-Moghadam. Optimized quantum circuit partitioning. *International Journal of Theoretical Physics*, 59(12):3804–3820, 2020.
- [6] Jens Eisert, Kurt Jacobs, Polykarpos Papadopoulos, and Martin B Plenio. Optimal local implementation of nonlocal quantum gates. *Physical Review A*, 62(5):052317, 2000.
- [7] Xiaojie Fan, Yukun Yang, Himanshu Gupta, and C. R. Ramakrishnan. Distribution and purification of entanglement states in quantum networks. *CoRR*, abs/2503.14712, 2025.
- [8] 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.
- [9] 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.
- [10] Mohammad Ghaderibaneh, Himanshu Gupta, C.R. Ramakrishnan, and Ertai Luo. Pre-distribution of entanglements in quantum networks. In *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 426–436, 2022.
- [11] 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.
- [12] 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.
- [13] 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, 2023.

- IEEE, 2023.
- [14] Eesa Nikahd, Naser Mohammadzadeh, Mehdi Sedighi, and Morteza Saheb Zamani. Automated window-based partitioning of quantum circuits. *Physica Scripta*, 96(3):035102, 2021.
 - [15] Nils Quetschlich et al. Mqt bench: Benchmarking software and design automation tools for quantum computing. *Quantum*, 2023.
 - [16] R. G Sundaram et al. Distributed quantum computation with minimum circuit execution time over quantum networks. In *IEEE QCE*, 2024.
 - [17] Ranjani Sundaram and Himanshu Gupta. Optimized generation of entanglement by real-time ordering of swapping operations. pages 1973–1979, 09 2024.
 - [18] 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.
 - [19] Ranjani G Sundaram and Himanshu Gupta. Dynamic distribution of quantum circuits with minimum execution time. In *2025 International Conference on Quantum Communications, Networking, and Computing (QCNC)*. IEEE, 2025.
 - [20] 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.
 - [21] Anocha Yimsiriwattana and Samuel J. Lomonaco. Generalized ghz states and distributed quantum computing. *arXiv: Quantum Physics*, 2004. URL: <https://api.semanticscholar.org/CorpusID:14682517>.
 - [22] Mariam Zomorodi-Moghadam, Mahboobeh Houshmand, and Monireh Houshmand. Optimizing teleportation cost in distributed quantum circuits. *International Journal of Theoretical Physics*, 57:848–861, 2018.