Distribution of Quantum Circuits Over General Quantum Networks

Ranjani G. Sundaram
Department of Computer Science
Stony Brook University
New York, USA

Himanshu Gupta
Department of Computer Science
Stony Brook University
New York, USA

C. R. Ramakrishnan
Department of Computer Science
Stony Brook University
New York, USA

***Abstract*—Near-term quantum computers can hold only a small number of qubits. One way to facilitate large-scale quantum computations is through a distributed network of quantum computers. In this work, we consider the problem of distributing quantum programs represented as quantum circuits across a quantum network of heterogeneous quantum computers, in a way that minimizes the overall communication cost required to execute the distributed circuit. We consider two ways of communicating: cat-entanglement that creates linked copies of qubits across pairs of computers, and teleportation. The heterogeneous computers impose constraints on cat-entanglement and teleportation operations that can be chosen by an algorithm. We first focus on a special case that only allows cat-entanglements and not teleportations for communication. We provide a two-step heuristic for solving this specialized setting: (i) finding an assignment of qubits to computers using Tabu search, and (ii) using an iterative greedy algorithm designed for a constrained version of the set cover problem to determine cat-entanglement operations required to execute gates locally.**

For the general case, which allows both forms of communication, we propose two algorithms that subdivide the quantum circuit into several portions and apply the heuristic for the specialized setting on each portion. Teleportations are then used to stitch together the solutions for each portion. Finally, we simulate our algorithms on a wide range of randomly generated quantum networks and circuits, and study the properties of their results with respect to several varying parameters.

I. Introduction

Motivation. There are two crucial technological hurdles that constrain the realization of quantum computing's potential: (a) the limited number of *qubits*, the basic store of quantum information, in any single quantum computer; (b) severe loss of information due to noisy operations and unwanted interactions with the environment [15]. The second hurdle can be overcome in principle by using error-correcting codes, but that results in a blowup in the number of qubits needed for a computation, thereby exacerbating the first hurdle. Distributing a quantum computation requiring a large number of qubits over a network of quantum computers (QCs) is a way to overcome this hurdle [5, 19, 20].

State of the Art. Quantum *circuits*, which specify a sequence of *gates* (operations) on a set of qubits, is a common abstraction between higher-level quantum *programs* and lower-level computing hardware. Distributing a quantum circuit over a quantum network involves assigning the circuit's qubits to QCs, and introducing communication operations to perform non-local operations (i.e. operations that span multiple QCs). The cost of distributing a circuit is the number of added communication operations.

Optimally distributing a given quantum circuit for evaluation over a network of QCs has been the focus of several earlier works [2, 6, 11]. They assume a *homogeneous* network— all QCs in the network have the same number of qubits, and the cost of quantum communication between any pair of QCs in the network is uniform. Even under this setting, the optimal distribution problem is intractable [2].

While [6] use teleportation as the only means of communication between QCs, [2] and [11] use *cat-entanglement* [19] which allows for the creation of shared copies of qubits (see §II) and often yields lower-cost solutions. But [2] and [11] ignore the storage requirements for multiple simultaneous cat-entanglements which could be substantial [2]. Distributed quantum computation over heterogeneous networks is considered in [10], where the cost, measured as increase in circuit depth, is not minimized, but bounded by a linear factor.

This Paper. In this work, we consider the problem of *optimally distributing a quantum circuit across an arbitrary topology network of heterogeneous quantum computers.* In particular, we consider the following generalizations to the optimal distribution problem:

1) The cost of communication between two QCs in a given network is a function of their network distance.
2) Each QC has specified qubit capacity, and an "execution memory" of limited size for storing cat-entangled qubits; these two limits may vary across QCs in the network.
3) Communication may be via cat-entanglement, or teleportation whose effect is to dynamically alter the assignment of qubits to QCs.

Performing distributed quantum computing in practice, when technology makes it feasible, will require us to drop the homogeneity assumption and study the problem in the more general setting described above.

Contributions and Organization. We formalize the optimal distribution problem under these generalizations as the DQC problem. We then provide polynomial-time heuristics for this intractable problem in multiple steps.

We first consider a special case of DQC, called DQC-M, that considers only cat-entanglement-based communication

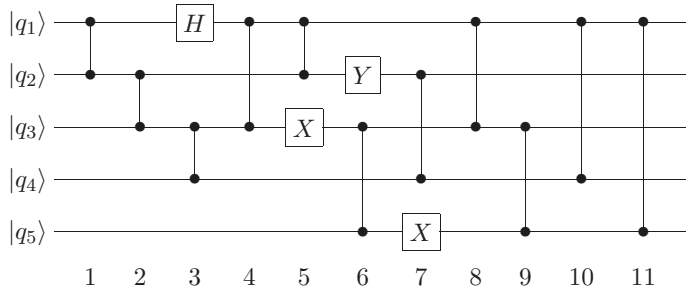

Fig. 1: Quantum Circuit Example.

(i.e., without generalization (3) above); see §IV. We solve DQC-M in two steps: (i) First, we use a Tabu-search-based heuristic to partition the given circuit's qubits among QCs taking the heterogeneity of the network and storage limits into account. (ii) Then, we develop an algorithm for introducing cat-entanglements to "cover" non-local gates, i.e., gates whose operands are in assigned to QCs. In a restricted setting, this yields an $\mathcal{O}(\log n)$-approximate solution (here n is the number of non-local gates).

For solving the general DQC problem (see §V), we provide two greedy heuristics, `Sequence` and `Split`, each using our solution to the DQC-M problem as a subroutine. Although `Sequence` has lower complexity than `Split`, neither is uniformly better than the other—we provide examples where each heuristic outperforms the other. In §VI, we present our evaluation results, which show that `Split` performs better than `Sequence` in most cases.

We begin with a brief background in quantum computation and communication (§II) for completeness followed by a formal description of the problem (§III).

II. Background: Quantum Computation and Communication

We start with giving a brief background on two key quantum concept relevant to our paper: quantum circuits and our choice of universal gate set as unary and CZ gates, and quantum communication methods used in our work.

Quantum Circuits. Quantum computation is typically abstracted as a *circuit*, where horizontal "wires" represent *qubits* which carry quantum data, and operations on the qubits performed by vertical "gates" connecting the operand wires [14]. Quantum computers (QCs) evaluate a circuit by applying the gates in the left-to-right order, so this circuit can also be understood as a sequence of machine-level instructions (gates) over fixed number of data cells (qubits).

Analogous to classical Boolean circuits, there are several universal gate sets for quantum computation: any quantum computation can be expressed by a circuit consisting only of gates from a universal gate set. In particular, the "*Controlled-Z*" binary gate, denoted by *CZ*, along with the set of all possible unary gates forms a universal gate set. We use this universal gate set in this paper since the symmetry of *CZ* gates allows a simpler formulation of distributed execution by creating linked copies using cat-entanglements (see below). Fig. 1 shows the pictorial representation of an example circuit, consisting only of unary gates (boxes) and *CZ* gates (vertical connectors). Without loss of generality, we ignore measurement gates; measurement can be postponed to the end and treated as unary operations.

Quantum Communication. If a given quantum circuit is to be evaluated in a distributed fashion over a network of QCs, we have to first distribute the qubits over the QCs. But such a distribution may induce gates in the circuit to span different QCs. To execute such *non-local* gates, we need to bring all operands' values into a single QC via quantum communication. However, direct/physical transmission of quantum data is subject to unrecoverable errors, as classical procedures such as amplified signals or re-transmission cannot be applied due to quantum no-cloning [8, 18].[1] Fortunately, there are other viable ways to communicate qubits across network nodes, as described below.

Teleportation. An alternative approach to physically transmit qubits is via *teleportation* [4] which requires an a priori distribution of maximally-entangled pair (MEP) of qubits (e.g., Bell Pair) over the two nodes. With an MEP distributed over nodes A and B, teleportation of a qubit state from A to B can be accomplished using classical communication and local gate operations, while consuming/destroying the MEP.

Cat-Entanglement: Creating "Linked Copies" of a Qubit. Another means of communicating qubit states is by creating *linked copies* of a qubit across QCs, via *cat-entanglement* operations [9, 20] which, like teleportation, require a Bell Pair to be shared *a priori*. These linked copies are particularly useful in efficient distributed evaluation of circuits involving only *CZ* and unary gates, as follows. The symmetry of *CZ* operation allows for either of the qubit operands to act as the (read only) control operand, and, more importantly, the control qubit can be just a linked copy of the original qubit operand (and since a linked copy is read-only, many copies can exist and used simultaneously). However, since a unary operation on the original qubit q may *change* its state, linked copies of q may not remain true copies; thus, we "disentangle" any linked copies of q via a dual operation called *cat-disentanglement* before applying a unary operation on q—the dis-entanglement operation doesn't require a Bell Pair.

III. Relevant Concepts and Problem Formulation

In this section, we define the DQC problem of distributing quantum circuits across quantum computers. We start with an informal description, define the relevant terms and concepts, and then formulated the DQC problem addressed in this paper.

Informal Problem Description. The goal of the *Distributing Quantum Circuits* (DQC) problem addressed in this paper is to determine an efficient distribution of a given quantum circuit, over a given network of QCs. Efficient distribution essentially entails two tasks: distributing the qubits over the distributed QCs, and then executing the given gates, including non-local

[1] Quantum error correction mechanisms [7, 13] can be used to mitigate the transmission errors, but their implementation is very challenging and is not expected to be used until later generations of quantum networks.

gates using a judicious combination of teleportation and/or cat-entanglement operations. Informally, the DQC problem is to execute the given (centralized) quantum circuit over the given quantum network using a minimum cost of teleportations and cat-entanglements used to execute the non-local gates, under the given memory constraints.

Closest Related Work. The closest work that addresses the above problem is our own recent work [11] —where we address the DQC problem under the simple settings of homogeneous computers with unbounded execution memory (to store cat-entanglement copies), complete network topology, and no teleportations. For the simplified setting, [11] presents a two-step algorithm for the DQC problem, wherein the first step determines the partitioning of qubits to computers through balanced graph partitioning and the second step minimizes the number of cat-entanglement operations via an iterative greedy approach.

In this paper, we address the generalized DQC problem wherein each computer may have non-uniform storage memory (to store the qubits) and bounded non-uniform execution memory (to allow for copies from cat-entanglements). Most importantly, we allow teleportations, which may dynamically change the partitioning of qubits across computers, but can improve the communication cost.

A. Key Concepts and Terminology

Quantum Circuit Representation. As in [2], we consider the universal gate set with (binary) *CZ* and unary gates. Also, in our context, we do not need to represent the type of unary gates. Thus, we represent an *abstract quantum circuit* C over a set of qubits $Q = \{q_1, q_2, \ldots\}$ as a sequence of gates $\langle g_1, g_2, \ldots\rangle$ where each g_k is either binary CZ gate or a unary gate. We thus represent binary gates in a circuit as triplets (q_i, q_j, k), where q_i and q_j are the two operands, and k is the time instant (see below) of the gate in the circuit; and unary gates as pairs (q_i, k), where q_i is the operand and k is the time instant. We use N_q and N_g to denote the number of qubits and gates in the circuit, respectively.

Each gate occurs uniquely at a time instant. In addition to the instants where the gates occur, we introduce additional **time instants** in between gates for cat-entanglement/teleportation operations. See Fig. 2.

Quantum Network (QN). We represent quantum network as a connected undirected graph with nodes representing QCs and edges representing (quantum and classical) direct communication links. Nodes of the network are denoted by P; we use the words node, computer, and QC interchangeably. We denote the number of nodes in the network by N_p. Each computer $p \in P$ has quantum memory to store qubits; for simplicity, we divide this memory into two parts: qubit storage memory to store the "original" qubits, with capacity denoted by s_p, and *execution memory* used to store the linked copies (ebits) from cat-entanglements, with capacity denoted by e_p. Thus, as part of the given QN specification, each node has has a certain amount of qubit storage and execution memory.

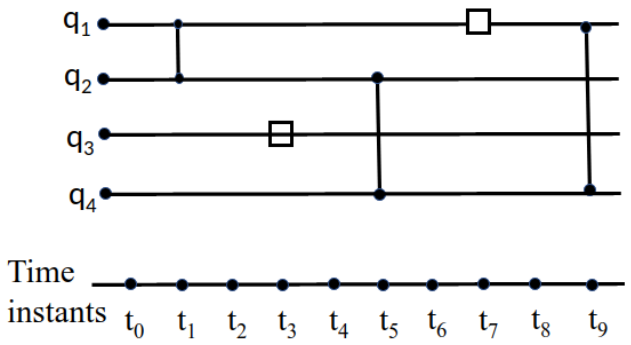

Fig. 2: In the above figure, the gates are at odd-numbered time instants $t_1, t_3, \ldots, t_9$, and the even-numbered time instants between each pair of consecutive gates have been introduced for convenience.

Home Computers. To distribute execution of a given quantum circuit, we first distribute qubits of the given circuit across the network nodes. At any point of time, each qubit q of the circuit resides (i.e., is stored) at a unique node in the network—which we call its *home computer* or just *home*. Cat-entanglement will create a linked copy of a qubit q at another computer, but does not change q's home. However, a qubit's home can be changed by teleporting it to another computer.

The home computers of qubits are represented by a set of home-computer functions, π_t, one for each time instant t. Each home-computer function maps a circuit's qubits to computers, i.e., $\pi_t : Q \mapsto P$. Thus, $\pi_t(q)$ denotes the home of qubit q at time t. A home-computer function π_t is valid if and only if it obeys the storage memory constraint— i.e., for any computer p with a storage memory of s_p units, there are at most s_p qubits q such that $\pi_t(q) = p$. A gate (q_i, q_j, t) is defined as **non-local** at time t if q_i and q_j have different home-computers, i.e., $\pi_t(q_i) \neq \pi_t(q_j)$.

Representing Teleportations. We represent teleportation by a triplet (q, p, t) which signifies that the qubit q was teleported to the computer p at time t. The teleportation (q, p, t) results in changing the home-computer function such that $\pi_t(q) = p$. For simplicity, we enforce that teleportations only happen at times with no gates. To simplify the issue of storage memory violations due to teleportations, we assume that all the teleportation at time t happen simultaneously- – and do not require additional memories for the EPs used in teleportations. Thus, the set of teleportations occurring at time t can be looked upon as changing the entire home-computer function from a valid π_{t-1} to a valid π_t.

Migrations (formalizing Cat-Entanglements). As described before, we use cat-entanglements to make linked copies of qubits to execute non-local gates. As in our earlier work [11], we use the term *migrations* to denote cat-entanglement. However, since we now allow teleportations which change the home computers of qubits over time, formal definition of migrations differs from that used in [11]. Informally, a qubit q can be migrated from its home computer to another for a certain duration of time (t_s, t_e); such a migration is considered

valid if, during the time interval (t_s, t_e), there are no unary operation on q and its home computer doesn't change. For simplicity, we assume that there are no gates at t_s and t_e.

Definition 1 (Migration): Given a quantum circuit, a quantum network, and the home-computer function at each time instant, a *migration* is a quadruple (q, p, t_s, t_e) to denote migration of qubit q from its home-computer at t_s to another computer p for the period (t_s, t_e). For the migration to be valid, the following conditions must hold.

- $p \neq \pi_{t_s}(q_i)$, i.e., q is migrated to a computer p different from its home computer at t_s.
- $\pi_t(q) = \pi_{t_s}(q)$ for all t in (t_s, t_e). That is, the qubit q's home computer doesn't change for the duration of the migration; i.e., q is not teleported during the period.
- There are no unary gates on q during (t_s, t_e). □

Coverage by a Migration; Home-Coverage. We use the term "cover" to denote migrations that help execute a non-local gate, and formally define the notion of coverage of a gate by migration(s) as follows. A binary gate $g = (q_i, q_j, t)$ can be covered by one or two migrations as follows.

1) By a single migration $(q_i, \pi_t(q_j), t_s, t_e)$ or $(q_j, \pi_t(q_i), t_s, t_e)$, where $t_s \leq t \leq t_e$; this represents migrating one operand to the other's home computer to enable the gate's local execution.
2) By a pair of migrations $\{(q_i, p, t_{s1}, t_{e1}), (q_j, p, t_{s2}, t_{e2})\}$ for some computer p, where $t_{s1} \leq t \leq t_{e1}$ and $t_{s2} \leq t \leq t_{e2}$. This represents migrating both operands to a common computer p and executing the gate locally there.

Coverage of a gate by a single migration is called *home-coverage*.

Feasible Set of Migrations. Limited execution memory at each computer restricts the maximum number of linked copies that can be present in a computer at any point of time. Consequently, a set of migrations is feasible only if the created linked copies obey the execution memory constraint at every computer at every point of time. We define this formally below.

Let m be a migration, p a computer, and t a time instant, and let $\mathcal{A}(m, p, t)$ be a function that is 1 iff there is a linked-copy of a qubit at computer p at time t due to migration m. More formally:

$$\mathcal{A}(m,p,t) = \begin{cases} 1 & \text{if } m = (q, p, t_s, t_e) \text{ and } t_s \leq t \leq t_e \\ 0 & \text{Otherwise} \end{cases}$$

A set of migrations $\mathcal{M}$ are said to be *feasible* if and only if $\sum_{m \in \mathcal{M}} \mathcal{A}(m, p, t) \leq e_p$ for all computers p, for all times t, where e_p is the capacity of execution memory at p.

Cost of Migrations and Teleportations. The cost of a migration (q, p, t_s, t_e) is defined as the distance between the nodes $\pi_{t_s}(q)$ to p in the given network graph. This cost accounts for the fact that migrating a qubit from $\pi_{t_s}(q)$ to p requires an EP over nodes p and $\pi_{t_s}(q)$ whose generation cost we assume to be proportional to the distance between p and $\pi_{t_s}(q)$. Similarly, cost of a teleportation (q, p, t) is defined as to be the distance between $\pi_{t-1}(q)$ to p.

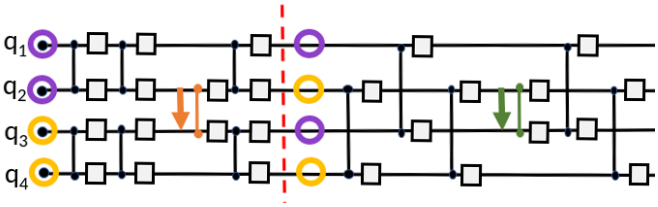

Fig. 3: DQC problem instance from Example 1

B. Problem Formulation and Example

We now define the DQC problem formally, based on the above concepts and terms.

DQC Problem. Given a quantum circuit and a quantum network, the DQC problem is to: (i) determine a valid home-computer function at all time instants (which also yields the teleportations incurred), and (ii) a feasible set of migrations that cover all the non-local gates, while minimizing the total cost of migrations and teleportations used.

The above DQC problem can be shown to be NP-hard, by a reduction from the DQC problem that only allows migration (and no teleportations) which is known to be NP-hard [2]. We omit the details of the reduction here.

Example 1. Consider a DQC problem instance in Figure 3—a circuit with four qubits and two computers each with a storage memory of two and execution-memory of 1. We assume the computers to be connected by a network link, and thus, the cost of any migration or teleportation is one. The figure also illustrates an optimal solution to the DQC problem of cost four. Initially, qubits q_1 and q_2 are assigned to the first computer (signified by purple circles), while q_3 and q_4 are assigned to the second computer (signified by yellow circles). At the time instant denoted by the red line, the qubit assignment is changed (via appropriate teleportations): q_2 is teleported to the second computer while q_3 is teleported to the first. The only non-local gates are the ones marked in orange and green — each requiring one migration. Thus, the total cost is 4, comprised of two teleportations and two migrations.

IV. DQC-M Problem: DQC with Only Migrations

For ease of presentation, we first address a simpler version of the DQC problem, wherein we do not allow any teleportations. We refer to this problem as DQC-M. In this special case DQC-M problem, the home-computers of the qubits never change after the initial placement and we need to cover all the non-local gates with just migrations. In effect, the DQC-M problem boils down to picking an initial assignment of qubits to computers such that non-local gates can be covered with minimum-cost migrations. We design a two-step algorithm for the DQC-M problem, as discussed below.

Two-Step Algorithm (`DQC-M`) for DQC-M. The DQC-M problem is a direct generalization of the problem addressed in our earlier work in [11] wherein the computers were assumed to have uniform storage memory with unbounded execution memory and the network was assumed to have a complete topology. As in [11], we develop a two-step algorithm, we call `DQC-M`, wherein in the first step we determine the assignment

Algorithm 1 DQC−M

Input: Quantum circuit C over qubits Q, Network graph G with nodes P
Output: A valid home-computer function π and a set of feasible migrations $\mathcal{M}^*$ that cover all non-local gates in C

1: $\pi \leftarrow$ A valid home-computer function such that the cost required migrations is low.
2: $\mathcal{M} \leftarrow$ A low-cost set of migrations that covers all the non-local gates resulting from π.
3: (Post-processing step) $\mathcal{M}^* \leftarrow$ A low-cost set of feasible migrations covering all non-local gates obtained by resolving memory constraint violations in $\mathcal{M}$.
4: **return** $\pi, \mathcal{M}^*$

of qubits to computers (i.e., the initial home-computer mapping which then remains unchanged), and then, in the second step, we determine the migrations to cover the non-local gates. We discuss these two steps in the following subsections. See the high-level pseudo-code of the two-step DQC−M algorithm.

A. DQC−M **Step 1: Assignment of Qubits to Computers**

Here, we address the first step of DQC−M— which assigns qubits to computers to minimize the cost of migrations required to cover all the non-local gates. In our earlier work [11] where we considered a special case of DQC-M problem with homogeneous network and unbounded execution memories, we used a balanced graph-partitioning to assign qubits to computers. However, in the current DQC-M problem, the cost of separating qubits q_1 and q_2 depends on the specific computers they are assigned to due to the network's heterogeneity— hence, a graph partitioning approach is inapplicable to the DQC-M problem's first step. Here, we develop a search-based algorithm— in particular, based on Tabu search [12]— to assign qubits to computers.

Tabu Search and Motivation. Tabu search is a local-search heuristic that starts with an initial solution, and then picks a better solution among the neighbors of the current solution. To avoid getting stuck in a local minimum, it sometimes also picks a worse solution, especially, if there is no better solution among the neighbors. The key distinction of Tabu search compared to other local-search algorithms is that it maintains a list of recently-visited solutions and incurs a penalty each time one of these solutions is chosen again. Our motivation for choosing a Tabu-based search heuristic is that our problem closely resembles the well-studied quadratic-assignment problem for which Tabu search has been shown to perform well [16]. This relationship is clear from our objective function shown under "Solution' Cost" below.

Tabu Search Algorithm for Assignment of Qubits. To design a Tabu-search based algorithm for our problem of assignment of qubits to computers, we need to define three key aspects of the algorithm: (i) Solution, (ii) Solution's neighbors, and (iii) Solution's Cost.

Solution and Its Neighbors. In our context, a solution is a valid home-computer function. Neighbors of a given solution π can be defined as valid solutions π' that result from either: (i) changing the assignment/mapping of a single qubit without violating the storage constraint, or (ii) "swapping" of two qubits mapped to two different computers in π.

Solution's Cost. A solution's cost can be defined as an estimate of the cost of migrations needed to cover the non-local gates resulting from the solution's qubit assignment. More formally, the cost of a solution π, denoted by $cost(\pi)$ is

$$cost(\pi) = \sum_{q_1, q_2 \in Q} w(q_1, q_2) \times \text{distance}(\pi(q_1), \pi(q_2))$$

where $w(q_1, q_2)$ is the *number* of migrations needed to cover the binary gates between q_1 and q_2 if they are assigned to different computers. We estimate $w(q_1, q_2)$ as described below.

Estimating $w(q_1, q_2)$. Let C is the original circuit that includes two qubits q_1 and q_2. To estimate $w(q_1, q_2)$ in C, we consider an induced circuit C' that consists only of qubits q_1 and q_2 and the sequence of gates from C that involve q_1 and q_2. We can compute the optimal number of migrations required to cover all the gates in the induced circuit C' when q_1 and q_2 are assigned to different computers using the *optimal* home-coverage algorithm (called MS-HC) from [11]. Note that in C' only home-coverage of gates by migrations is possible. We use the optimal number of migrations required in C' as the estimate for $w(q_1, q_2)$ in the given circuit C.

Tabu Algorithm. Based on the above discussion, the algorithm (called Tabu) for the first step of DQC−M is defined as follows:

1) $\pi^* = \pi =$ initial random solution
2) $L =$ [ ] /* a bounded-length list of forbidden solutions */
3) Repeat for λ iterations:
   a) $\pi = argmin_{\pi' \in neighbors(\pi) - L}\ cost(\pi')$
   b) $\pi^* = \pi$ if $cost(\pi) < cost(\pi^*)$
   c) $L = L \cup \{\pi\}$, removing the oldest element from L if necessary to maintain length bound.
4) Return π^*

B. DQC−M **Step 2: Selection of Migrations to Cover Gates**

In this section, given an assignment of qubits to computers, we seek to compute a minimum-cost feasible set of migrations that cover all the non-local gates.

Basic Idea. Selecting migrations to cover non-local gates is essentially a generalization of the set-cover problem, with two key differences:

1) First, we are restricted to choose only a feasible set of migrations. Fortunately, the execution-memory constraints can be expressed as linear constraints—and hence, can be handled by using approximation techniques from [3] that studies the related maximum-coverage problem with linear constraints.
2) Second, a gate may be covered by a pair of migrations together which translates to allowing a pair of sets to cover

Algorithm 2 Step 2 of DQC−M.

Input: Quantum Network G, Quantum circuit C, Home-computer function π.
Output: A set of migrations $\mathcal{M}$ that covers all non-local binary gates.

uncovered ← non-local binary gates in C due to π.
while (uncovered) **do**
    (S, covered) ← COVER-α(uncovered, G, π)
    uncovered ← uncovered \ covered
    $\mathcal{M} \leftarrow \mathcal{M} \cup S$
return $\mathcal{M}$.
function COVER-α(uncovered, G, π)
    minCost = 1;
    maxCost = |uncovered| × (Diameter of G)
    for c in [minCost, maxCost] **do**
        (S, covered) ← AG−Algo (c, C, π, G, uncovered)
        if |covered| ≥ α|uncovered| **then**
            Break
    return (S, covered)

an element together; see the "*Coverage by a Migration*" paragraph in §III-A. Such a generalization breaks the submodularity of the objective function—and in general, can render the coverage problem inapproximable.

It should be noted that with generalization (2) alone but in the absence of (1), we gave an approximation algorithm in [11]. However, neither the technique from [3] nor [11] can be extended to handle both the above generalizations together while ensuring a performance guarantee. Thus, we develop a heuristic based on [3] as described below. We start with considering the special case of home-coverage (i.e., select migrations to cover gates using only home-coverage), and then extend our algorithm to general coverage.

Approximation Algorithm for Home-Coverage. We note that MULTIPLICATIVE-UPDATES algorithm from [3], hereafter referred to as the AG−Algo, maximizes the number of elements covered under a *given cost budget* of sets with linear constraints. In contrast, the Step-2 of DQC−M needs to select minimum-cost migrations to cover *all* gates—which is in some sense, a dual of the problem solved by AG−Algo. To cover all the gates with minimum-cost migrations based on AG−Algo, we use an iterative algorithm where, in each iteration, we cover at least a certain constant fraction α of the remaining gates using a minimum-cost set of migrations. Iterations are repeated until all binary gates are covered. To find a minimum-cost set of migrations covering at least α fraction of the gates using AG−Algo, we exploit the fact that cost is an integer bounded by the product of the number of binary gates in a circuit and the diameter of the network, as shown below.

- For each cost c:
  - Select, using AG−Algo, a feasible set of migrations costing at most c that covers the maximum number of gates.
- Pick the solution with smallest c for which the AG−Algo solution could cover α fraction of the remaining gates.

We use $\alpha = 0.4$ in our implementation, based on the approximation factor of the AG−Algo. For a more formal and complete description, see the pseudocode shown in Algorithm 2. We make two remarks. First, while each iteration returns a feasible set of migrations, their union may not be feasible, i.e., may violate execution-memory constraints; we resolve them as a post-processing step below. Second, in subroutine COVER-α, we can use a binary search to more efficiently iterative over all possible costs.

For sake to clarity, we have intentionally omitted details of the AG−Algo algorithm, but at a high-level it is an iterative approach that picks the migration based on an objective that considers both—the number of gates covered as well as the ability to cause constraint violations.

Performance Guarantee. It can be shown that the above algorithm yields a $\mathcal{O}(\log n)$-approximation solution (where n is the number of non-local gates) for the problem of selection of minimum-cost set of migrations to cover all gates given a home-computer function, while bounding the violation of execution-memory constraints (violations fixed in §IV-C). Note that $n \ll N_g$, the number of all gates in the circuit. We formalize the performance guarantee below.

Theorem 1: Given a network G, circuit C, and a home-computer function π, let k^* be the optimal-cost of a set of feasible migrations that home-covers all the non-local gates for π, and n be the total number of non-local gates for π. Algorithm 2 returns a solution $\mathcal{M}$ such that:

- $\mathcal{M}$ covers all non-local gates.
- $|\mathcal{M}| \leq (\log n)k^*$.
- For every computer p in G, the amount of execution memory in p used by $\mathcal{M}$ at any time instant is at most $(\log n)e_p$. ■

Generalization to General Coverage. Recall that the above algorithm was under the restriction of home-coverage. To allow for general coverage of gates by migrations, i.e., to allow a pair of migrations to together cover a gate, we need to modify the AG−Algo subroutine accordingly. Note that AG−Algo works iteratively, wherein in each iteration it selects a single migration. To allow for general coverage, we modify the AG−Algo subroutine to also consider pairs of migrations for selection in each iteration. This change is straightforward, and we omit the details. Unfortunately, allowing general coverage by migrations breaks down the approximation guarantee of AG−Algo.

C. Post-Processing to Resolve any Memory Violations.

While our algorithm selects a feasible set of migrations in every iteration, the overall solution may violate memory constraints. We resolve these violations by replacing some migrations with migrations of smaller duration. Consider a migration (q, p, t_s, t_e) which covers gates at time t_1, t_2 and t_3. We can covert this migration into three separate migrations, viz., $(q, p, t_1, t_1), (q, p, t_2, t_2)$ and (q, p, t_3, t_3), each of which

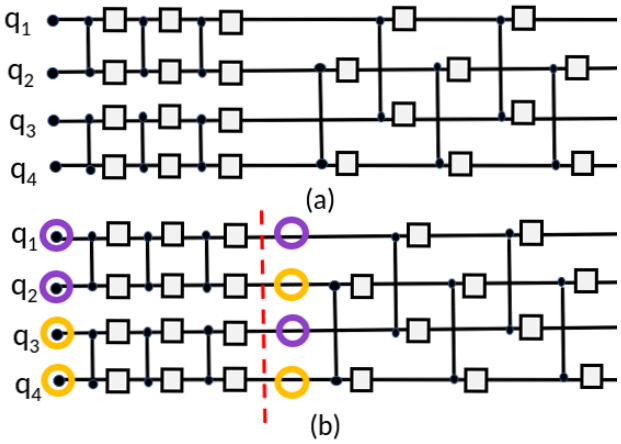

Fig. 4: Circuit illustrating the benefit of including teleportations, see Example 2.

covers the gates at t_1, t_2 and t_3 respectively. The above conversion reduces the usage of execution memory at p, while increasing the total cost of migrations. Note that there always exists a solution that uses only such "instanteneous" migrations and requires only one unit of execution memory at each computer, since there is at most one gate at each time instant. Thus, a simple strategy to resolve execution-memory violations could be to convert migrations into multiple shorter migrations iteratively.[2]

Based on the above, our post-processing algorithm to resolve execution-memory violations is as follows: we iteratively pick the migration that causes a violation and covers the least number of gates, and covert it into instantaneous migrations as described above.

Time Complexity of `DQC-M` Algorithm. Overall, the `DQC-M` Algorithm runs in $\mathcal{O}(\lambda {N_q}^3 + N_p n^6 \log n)$ time, where λ is the number of iterations chosen for our Tabu search heuristic,N_q is the number of qubits, N_p is the number of computers in the quantum network, and n is the number of non-local binary gates with the chosen home-computer function; note $n \ll N_g$, the number of gates in the circuit. In our implementation, we pick λ to be 20, beyond which Tabu search offers minimal improvement in solutions for our instances.

V. General DQC Problem (with Teleportations)

We now consider the general DQC problem which allows teleportations as well as migrations. We start with illustrating the benefit of teleportations. Then, we design two algorithms for the general DQC problem; our algorithms use `DQC-M` from the previous section as a subroutine.

Example 2. Benefit of Teleportations. Consider the circuit instance shown in Fig. 4(a). We seek to distribute this circuit across two computers, each with a storage memory of two units. If we allow only migrations, then it is easy to observe that the optimal solution assigns the qubits $\{q_1, q_2\}$ and $\{q_3, q_4\}$ to the two computers respectively and uses five migrations to cover all the gates (since only 5 of the 11 gates are non-local for the given home-computer mapping). Now, consider the solution shown in Fig. 4(b) which uses teleportations too. Here, we essentially change the home-computer mapping at the time instant denoted by the red vertical line, which makes all the gates local; thus, the total cost is only 2—for the two teleportations needed to change the home-computer function at the red line. Note that the above example can be scaled to exhibit arbitrarily large the benefit from using teleportations.

[2]Note that since our algorithms only create migrations for yet-uncovered gates, such a conversion strategy would not yield "redundant" instantaneous migrations—and thus, conversions alone should yield a feasible solution.

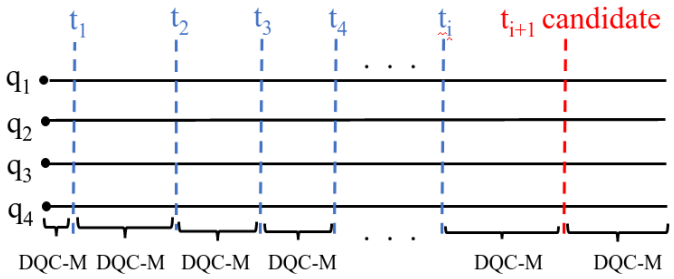

Fig. 5: `Sequence` algorithm at index t_{i+1}. Note that the points of teleportation are determined in sequence from right to left.

`Sequence` **Algorithm.** Our first algorithm called `Sequence` is a greedy approach, wherein we go over the given circuit from left to right, and determine, at each gate, whether or not changing the home-computer function just before it would be beneficial to the overall cost of teleportations and migrations required. Recall that we create additional time instants in between gates, and teleportation can only happen at these additional non-gate instants. Consider a binary gate at time $t+1$. Let us assume that the `Sequence` algorithm has already determined the teleportation points for all the time points before $t-1$. To determine whether teleportations should happen at t (i.e., the home-computer function should be changed at t), we estimate the total cost incurred for the full circuit with optimal teleportations at t (and none later than t) and compare this estimated cost with no teleportations at t or later. The cost incurred for the full circuit with optimal teleportations at t, with i prior instants where teleportations were done, can be estimated in the following way.

(a) Run the `DQC-M` algorithm on each of the $i+2$ sub circuits resulting from the i previously chosen instants of teleportation and t.
(b) Compute the total migration cost by adding the migration cost of each sub circuit.
(c) Compute the total teleportation cost by adding the teleportation cost between every pair of consecutive sub circuits.
(d) The cost incurred for the full circuit with optimal teleportations at t is the sum of the total migration cost and total teleportation cost.

The total cost of the whole circuit without teleportation

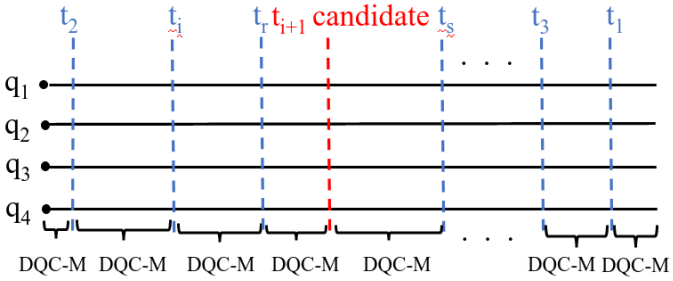

Fig. 6: `Split` algorithm's iteration considering t_{i+1} as the next teleportation-point. Note that the points of teleporation may not be determined in sequence.

at t_{i+1} can also be similarly estimated (in fact, has already been computed in previous iterations of the algorithm). If the cost with teleportations at t is lower than without the teleportations at t, then t is added to the list of time instants where teleportation is to be done.

`Split` **Algorithm.** Our alternate approach to solving the DQC problem is the `Split` algorithm. `Split` is similar to `Sequence` in that `Split` also select points of teleportations iterations through a similar cost-estimation methodology. However, rather than going over the circuit from left to right, `Split` iteratively picks the best time instant *anywhere* in the circuit where teleportation will help the most. Consider a stage in the algorithm, where the time instants $t_1, t_2, \ldots, t_i$ have already been determined to be points of teleportations in previous iterations; note that these points need not be ordered left to right. Then, in the following iteration, the algorithm determines the next point t_{i+1} of teleportation; this determination is done by exhaustive search, by considering all possible points in the circuit and picking the one that yields the best total cost estimate. The total cost can be estimated in a similar manner as described in the previous `Sequence` algorithm. A high-level pseudo-code of SPLIT is as follows.

- Assume i points of teleportations $t_1, t_2, \ldots, t_i$ have already been chosen; these may not be in left-to-right order. We describe how to select t_{i+1}.
- For every possible time instant t in the circuit not in $\{t_1, t_2, \ldots t_i\}$:
  - Note that choosing t as a point of teleportation results in $i + 2$ sub-circuits.
  - Run `DQC-M` on each of these sub-circuits to obtain an initial home-computer function and a migration cost associated with each sub-circuit.
  - Determine the total estimated cost of picking $t = \sum$ migration costs + $\sum$ teleportation cost of changing the home-computer function.
- Pick the t with minimum cost as t_{i+1} if and only if it reduces cost from the previous iteration (cost associated with t_i).
- Repeat the above steps until no t reduces the cost.
- Run `DQC-M` on each sub-circuit to obtain an initial qubit assignment, a set of migrations and a set of teleportations.

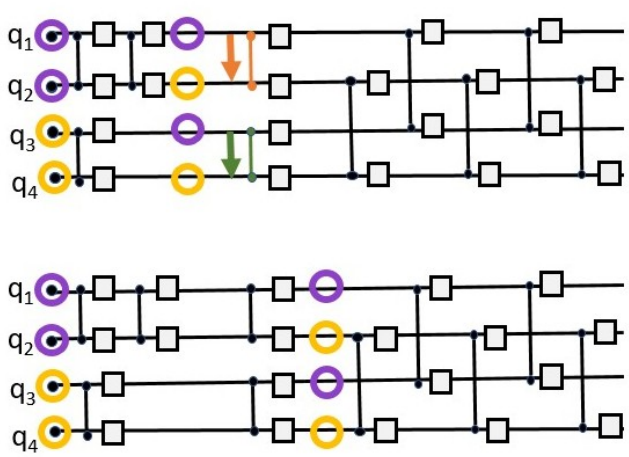

Fig. 7: An example wherein `Split` outperforms `Sequence`.

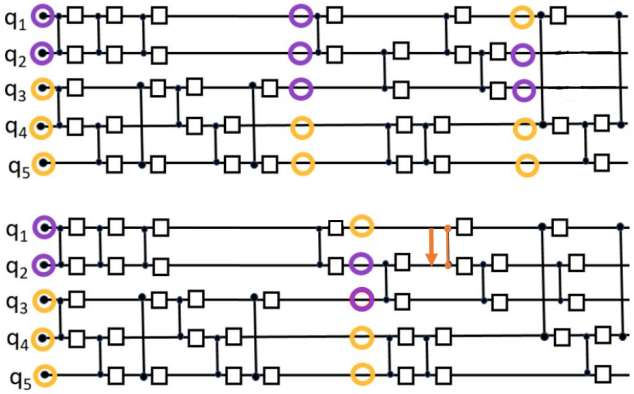

Fig. 8: An example wherein `Sequence` outperforms `Split`.

As in the `Sequence` algorithm, we note that many of the cost components remain unchanged (as the sub-circuits remain unchanged) from one iteration to the next. Hence, these cost components need not be recomputed.

`Sequence` **vs.** `Split` **Algorithms.** Since `Split`'s search methodology is more general, it is expected to outperform `Sequence`; this observation is also confirmed in our empirical results in the next section. However, there are specific instances of the DQC problem where either may outperform the other.

Fig. 7 shows an instance where `Split` outperforms `Sequence` and Fig. 8) shows an instance where `Sequence` outperforms `Split`. In Fig. 7, we observe that `Sequence` tends to pick the earliest point of teleportation that offers an improvement in cost. In contrast, `Split` parses the entire circuit and picks the best point of teleportation. This, however, is not always preferable. For example, in Fig. 8, `Split` chooses the best point of teleportation in the first iteration (by arbitrary tie-breaking). This forces `Split` to perform one migration and two teleportations. In subsequent iterations, `Split` can at best replace the migration with a teleportation,

and can never reduce the cost.

Time Complexity of `Sequence` and `Split`. `Sequence` algorithm parses the circuit from left to right, deciding whether or not to teleport at each time instant. Its time complexity is $\mathcal{O}(N_g(T_{DQC\text{-}M} + N_qN_p))$, where N_q is the number of qubits, N_p is the number of computers, N_g is the number of gates in the circuit, and $T_{DQC\text{-}M}$ is the time taken by the `DQC-M` algorithm. `Split` Algorithm repeatedly parses the whole circuit and picks one point of teleportation in each parse. Its time complexity is $\mathcal{O}({N_g}^2(T_{DQC\text{-}M} + N_qN_p))$. In both cases, the term N_qN_p arises from the computation of teleportations.

VI. Evaluation

Here, we present the evaluation of our algorithms over randomly generated quantum circuits and networks. The performance metric used in our evaluations is the the overall communication cost comprising of the total cost of migrations and teleportations as defined before.

Algorithms Compared. We compare the following four algorithms: (i) `DQC-M` from §IV, which uses only migrations; (iii) `DQC-M-Greedy` which is same as `DQC-M`, except that it uses a simple greedy algorithm[3] instead of `DQC-M`'s Step 2 to select migrations; (iii) `Sequence` from §V which determines teleportation points by scanning the circuit from left to right; (iv) `Split` from §V which determines teleportations iteratively at arbitrary time instants.

Generating Random Quantum Networks. A quantum network is created based on the following parameters.

- Number of quantum computers
- Probability of a link, between a pair of nodes.
- Qubit storage capacity of each computer
- Execution memory capacity of each computer

To ensure that we generate only connected networks, we use a Python-based library [1] to repeatedly generate Erdős-Rényi graphs with a given edge probability until a connected graph is obtained. Erdős-Rényi graphs are generated on k vertices by choosing every edge uniformly at random with probability p. Erdős-Rényi have the useful property that when $p > \frac{(1+\epsilon)\log k}{k}$ for some small $\epsilon > 0$, the generated graph is connected with high probability. In our case, the choices of k and p are sufficient to ensure high probability of connectivity. We choose the qubit storage capacity of each computer to be 60% to 140% of the average storage requirement (ratio of number of qubits to number of computers). Similarly, we choose the execution memory capacity for each computer to be 30% to 70% of the average storage requirement.

Generating Random Quantum Circuits. A quantum circuit is created based on the following parameters.

- Number of qubits
- Total number of gates (unary and binary) per qubit
- Fraction of binary gates, i.e., ratio of binary gates to the total number gates

Let f be the fraction of binary gates. We generate the gates sequentially, and, at each point, determine whether the gate should be binary (unary) with probability of f $(1 - f)$. Then, we choose the gate operand(s) randomly.

Evaluation Results. We evaluate each of the above four algorithm over generated random networks and circuits as described above. We vary six parameters in our simulations (the parenthesized values are the corresponding default values): (i) number of computers (10); (ii) probability of an edge (0.5); (iii) number of qubits (50); (iv) number of gates per qubit (50); (v) fraction of binary gates (0.5), (vi) execution memory capacity (30% to 70% of the average storage requirement).

In each of the experiments, we vary one of the above five parameter values, while fixing the remaining five to their default values. We present the evaluation plots in Figures 9a-9e.

Overall, we make the following observations on the relative performances of the algorithms compared.

- Using a combination of teleportations and migrations offers a significant reduction in cost compared to using only migrations; this is by observing that `Sequence` and `Split` algorithms outperform the other two algorithms in all of our experiments. In particular, in Fig. 9a we see that allowing teleportations reduces the total cost by around 10% on average using `Sequence` and around 15% using `Split` algorithm.
- In almost all cases, `Split` outperforms `Sequence`; this is as expected, since `Split` can be looked upon as a generalization of the `Sequence` algorithm in terms of the candidates considered for teleportation times.
- The order of the algorithms from highest to lowest performance is: `Split`, `Sequence`, `DQC-M`, `DQC-M-Greedy`; as expected, `DQC-M-Greedy` performs the worst since it completely disregards the execution-memory constraints initially and resolves the resulting violations in post-processing.

We also observed that the `DQC-M` algorithm rarely resulted in memory violations, and thus, rarely required any post-processing.

We also make the following observations regarding how the performance of the algorithms varies with varying parameter values. In Fig. 9c, we observe that with the increase in the ratio of binary gates, the performance gap between the various algorithms decreases—since with fewer unary gates, migrations don't need to be disentangled much which allows them to cover more gates, reducing the advantage of teleportations over migrations. In Fig. 9d, we observe that the cost of all algorithms decrease with increasing probability of a network edge, due to shorter paths. In Fig. 9e, we observe that the cost of all algorithms increase as expected with increase in total number of gates. Finally, in Fig. 9f, we vary the execution memory capacity of each computer in the quantum network from 1 to 6 units, that is, 20% to 120% of

[3] An iterative greedy algorithm that selects either a single migration or a pair of migrations that cover the most number of uncovered binary gates without considering execution memory constraints.

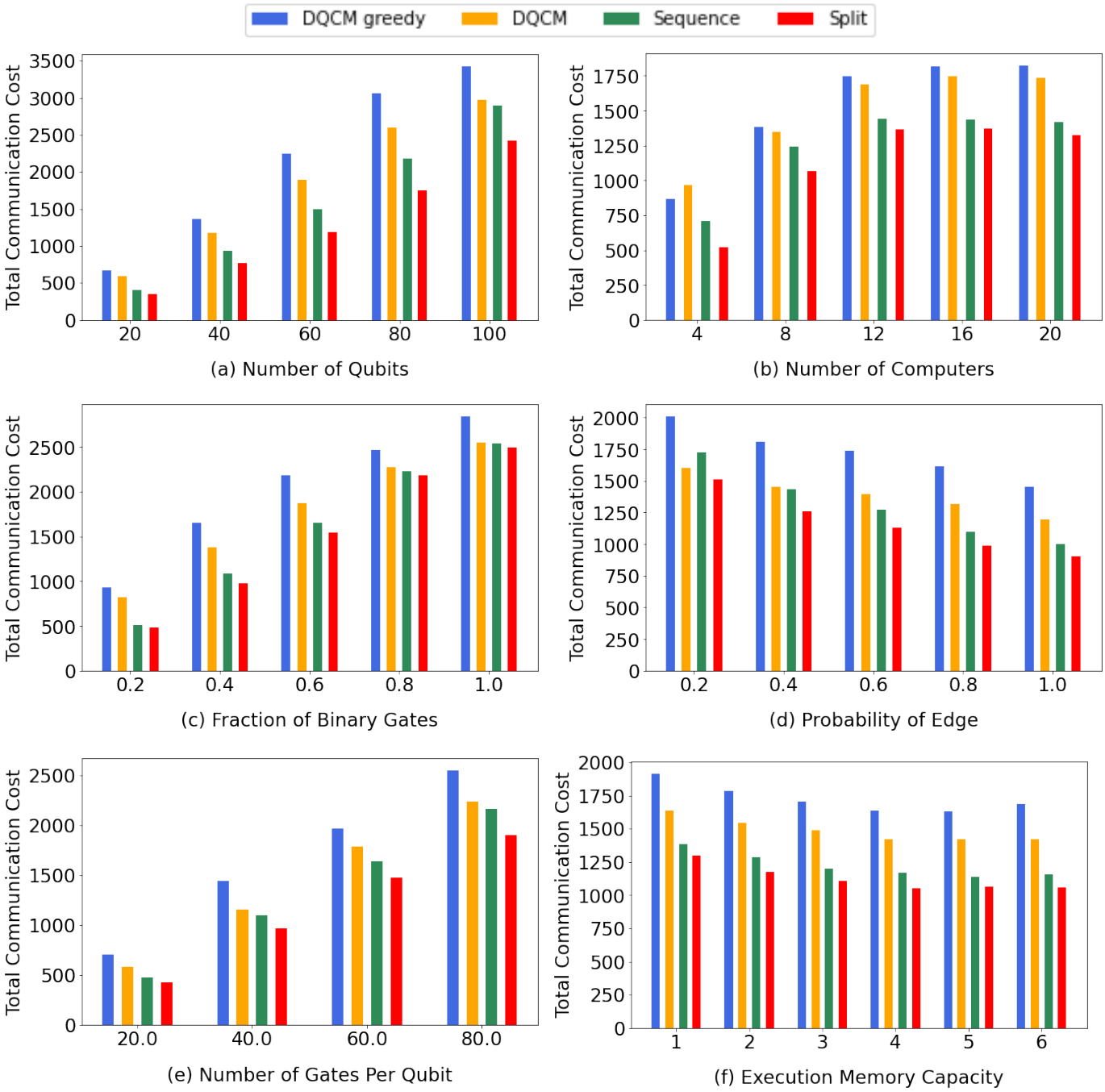

Fig. 9: Total communication cost incurred by different algorithms for varying parameter values.

average storage requirement (note that the execution memory capacity per computer used in all other plots is 30% to 70% of average storage requirement which amounts to 1 to 4 units for each computer). As expected, cost decreases with increasing execution memory until there is sufficient memory.

VII. Conclusion

In this paper, we consider the problem of distributing a quantum circuit across a network of heterogeneous quantum computers in a way that minimizes the overall communication cost. We described efficient algorithms which tackle issues that arise due to the heterogeneity of the network and different modes of communication. We evaluated our algorithms on randomly-generated quantum circuits and networks to study their performance. Several avenues of future research remain. Efficient simulation of quantum computations on classical machines (e.g., [17]), and analysis of quantum programs (e.g., [21]) employ partitioning similar to this work; it will be interesting to investigate this relationship further.

REFERENCES

[1] Networkx. https://networkx.org/documentation/stable/reference/generated/networkx.generators.random_graphs.erdos_renyi_graph.html.

[2] P. Andres-Martinez and C. Heunen. Automated distribution of quantum circuits via hypergraph partitioning. *Phys. Rev. A*, 100(3), 2019.

[3] Yossi Azar and Iftah Gamzu. Efficient submodular function maximization under linear packing constraints. In *International Colloquium on Automata, Languages, and Programming*, pages 38–50. Springer, 2012.

[4] C. H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W. K. Wootters. Teleporting an unknown quantum state via dual classical and Einstein–Podolsky–Rosen channels. *Phys. Rev. Lett.*, 70(13), 1993.

[5] JI Cirac, AK Ekert, SF Huelga, and Chiara Macchiavello. Distributed quantum computation over noisy channels. *Physical Review A*, 59(6):4249, 1999.

[6] O. Daei, K. Navi, and M. Zomorodi-Moghadam. Optimized quantum circuit partitioning. *Int. J. Theor. Phys.*, 59(12):3804–3820, 2020.

[7] Simon J Devitt, William J Munro, and Kae Nemoto. Quantum error correction for beginners. *Reports on Progress in Physics*, 76(7):076001, 2013.

[8] D. Dieks. Communication by EPR devices. *Physics Letters A*, 92(6), November 1982.

[9] J. Eisert, K. Jacobs, P. Papadopoulos, and M.B. Plenio. Optimal local implementation of non-local quantum gates. *Phys. Rev. A*, 62(052317-1), 2000.

[10] 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. https://doi.org/10.1109/TQE.2021.3053921 doi:10.1109/TQE.2021.3053921.

[11] 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.

[12] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549, 1986. Applications of Integer Programming.

[13] Sreraman Muralidharan, Linshu Li, Jungsang Kim, Norbert Lütkenhaus, Mikhail D Lukin, and Liang Jiang. Optimal architectures for long distance quantum communication. *Scientific reports*, 6(1):1–10, 2016.

[14] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[15] John Preskill. Quantum Computing in the NISQ era and beyond. *Quantum*, 2:79, August 2018.

[16] Jadranka Skorin-Kapov. Tabu search applied to the quadratic assignment problem. *ORSA Journal on computing*, 2(1):33–45, 1990.

[17] Zhimin Wang, Zhaoyun Chen, Shengbin Wang, Wendong Li, Yongjian Gu, Guoping Guo, and Zhiqiang Wei. A quantum circuit simulator and its applications on Sunway TaihuLight supercomputer. *Scientific Reports*.

[18] W. K. Wootters and W. H. Zurek. A single quantum cannot be cloned. *Nature*, 299(5886), October 1982.

[19] A. Yimsiriwattana and S. J. Lomonaco Jr. Distributed quantum computing: A distributed Shor algorithm. *Quantum Information and Computation II*, 5436, 2004.

[20] A. Yimsiriwattana and S. J. Lomonaco Jr. Generalized GHZ states and distributed quantum computing. *AMS Cont. Math.*, 381(131), 2005.

[21] N. Yu and J. Palsberg. Quantum abstract interpretation. In *42nd ACM SIGPLAN Conference on Programming Language Design and Implementation,*.