Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/TQE.2020.DOI

DQRA: Deep Quantum Routing Agent for Entanglement Routing in Quantum Networks

LINH LE and TU N. NGUYEN (SENIOR MEMBER, IEEE)

College of Computing and Software Engineering, Kennesaw State University, Marietta, GA 30060, USA (e-mail: {linh.le and tu.nguyen}@kennesaw.edu).

Corresponding author: Tu N. Nguyen (email: tu.nguyen@kennesaw.edu).

This work was supported in part by the National Science Foundation under Grant CNS-2103405.

ABSTRACT Quantum routing plays a key role in the development of the next-generation network system. In particular, an entangled routing path can be constructed with the help of quantum entanglement and swapping among particles (e.g., photons) associated with nodes in the network. From another side of computing, machine learning has achieved numerous breakthrough successes in various application domains including networking. Despite its advantages and capabilities, machine learning is not as much utilized in quantum networking as in other areas. To bridge this gap, in this paper we propose *a novel quantum routing model* for quantum networks that employs machine learning architectures to construct the routing path for the maximum number of demands (source-destination pairs) within a time window. Specifically, we present a deep reinforcement routing scheme that is called Deep Quantum Routing Agent (DQRA). In short, DQRA utilizes an empirically designed deep neural network that observes the current network states to accommodate the network's demands which are then connected by a qubit-preserved shortest path algorithm. The training process of DQRA is guided by a reward function that aims toward maximizing the number of accommodated requests in each routing window. Our experiment study shows that, on average, DQRA is able to maintain a rate of successfully routed requests at above 80% in a qubit-limited grid network, and approximately 60% in extreme conditions i.e. each node can be repeater exactly once in a window. Furthermore, we show that the model complexity and the computational time of DQRA are polynomial in terms of the sizes of the quantum networks.

INDEX TERMS quantum network routing; deep reinforcement learning; quantum networks; next-generation network; deep learning; machine learning.

I. INTRODUCTION

There are high demands of network resources and security in today's network and the next-generation network systems since more and more devices are connected to the Internet and new services are created. Quantum network appears as a promising technology to enhance exchanged information security via the Internet [1], [2], [3], [4]. With recent advances in quantum computing technology [5], [6], [7], [8], a quantum network is built upon the conventional network (e.g., network slicing) that is composed by various nodes (computers) equipped with quantum processors to process and deliver information in the form of quantum bits, called *qubits* [1], [2], [9], [10].

Quantum networks are not designed to replaced the conventional network communication. In fact, they supplement the operation of the next-generation network system where quantum entanglement and swapping play the key role of quantum network technology. In particular, quantum entanglement is designed with the no-cloning theorem, in which it is impossible to produce independent and identical copies of any unknown quantum states. This addresses the fundamental problem of network security: *key distribution* [11], [12]. Specifically, quantum entanglement is set up based on a strong correlation between two particles (i.e., photons). Hence, quantum entanglement can enable the secured data transmission, called *teleportation*, as shown in Fig. 1. A routing path in the quantum network is therefore built based on quantum entanglement as well with the support of quan-

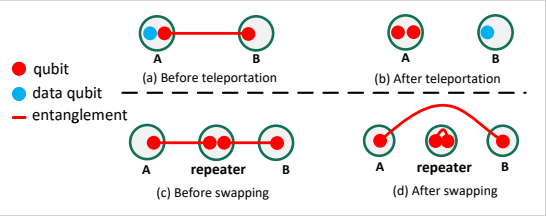


FIGURE 1: Transmission of qubit using teleportation in (a-b) and quantum entanglement swapping in (c-d).

tum repeater using swapping protocol to enable entanglement to be distributed over long distances [13]. We will present details of quantum repeater and swapping in §II-A. Since quantum routing is completely relied on the qubit entanglement, the question of how to construct routing paths for a (or multiple) given pair(s) of source and destination becomes how to assign qubit entanglements to appropriate routing paths. In order to design scalable quantum networks, existing works aim at proposing new optimization models to efficiently assign qubits and repeaters to different quantum entanglements. The authors in [14] study the method to apply Dijkstra algorithm to find the shortest path for repeater assignments. On the other hand, the authors in [15] discuss how to use a limited number of repeater to enable quantum communication [16] and propose multi-path routing in a diamond topology.

While machine learning has been a highly active research area recently, works on machine-learning-based routing in quantum network are still relatively limited at this moment. Authors in [17] propose a model that utilizes reinforcement learning to schedule entanglements. Specifically, the work in [17], focuses on optimizing entanglement times on quantum channels across a path to ensure an entanglement state between the two end nodes can be established before any channels decay. In this paper, we focus on using machine learning on a different aspect of routing, which is to allocate quantum channels to accommodate multiple communication requests in a quantum network. To our knowledge, there are currently no such works in the literature. With such motivation, in this paper, we present a deep reinforcement routing scheme that is called *Deep Quantum Routing Agent (DQRA)*. We start with modeling the problem of entanglement routing as a reinforcement learning problem with the following settings:

- **Network environment** includes information on the current network graph (i.e. which nodes can establish entanglement with which others), qubit capacity at each node, and a set of pending routing requests (demands).
- **Episodes** are the accommodation of all requests in a given window. An episode ends by solving all requests in a set, or by entering an environment state in which no pending requests can be solved. Each step in an episode refers to the accommodation of one request.
- **Actions** that the agent makes at each step include selecting then routing a pending request.
- **Rewards** that the agent receives after an action are designed to guide the agent to generate schedule that maximize the number of accommodated requests in a given time window.

DQRA then solves the problem using a combination of a deep neural network and a shortest path algorithm. The neural network first observes the current environment state to select a pending request to accommodate. The selected request is then routed using a shortest path algorithm that uses a metric representing the qubit capacities among nodes across a path. We use two algorithms to train the deep network of DQRA, as a deep reward network, and as a deep Q network (DQN) [18]. In the first case, the neural network predicts the true reward obtained if a request is selected, and in the second case, the neural network predicts the Q-values of selecting the requests. Both training algorithms are guided by the previously mentioned reward function. Our experiment study shows that, on average, both DQRA models are able to maintain a rate of successfully routed requests at 85–90% in a qubit-limited quantum network, and 60–75% in extreme conditions of network (i.e. all nodes can only be end nodes or repeater exactly once). We also empirically show that the routing time of DQRA increase as a polynomial function of network sizes (i.e. number of nodes). Specifically, the paper has the following **contributions** and **intellectual merits**:

1) We tackle the problem of entanglement routing in quantum networks from a machine learning perspective. Specifically, we propose a reinforcement learning model for the problem with specific designs of environments, actions, and rewards, that guide the agents towards a schedule that fulfills the most traffic requests.
2) We present DQRA, a deep reinforcement routing scheme that consists of an empirically designed deep neural network that schedules requests and a qubit-preserved shortest path algorithm that routes selected ones. The deep network of DQRA can be trained either as a deep reward network or a deep Q network. DQRA is shown to obtain good request-resolving rates even in qubit-scarce networks, and is scalable in terms of model complexity and routing times.

Organization: The rest of the paper is organized as follows. Section §II discusses the definitions and notations that related to our work, and mathematically formalizes our research problem. Section §III reviews the related concepts in deep reinforcement learning. We discuss DQRA in details in Section §IV, and present the experiment study in Section §V. Finally, we conclude our paper in Section §VI.

II. NETWORK MODEL AND RESEARCH PROBLEM

In the following sections, an overview of the quantum network and basic mathematical notations are initially introduced. We also illustrate simple examples and formally define the research problem to present the key ideas behind the proposed model and methodology.

IEEE Transactions on Quantum Engineering

A. QUANTUM NETWORK: AN OVERVIEW

We first briefly introduce basic definitions in the context of a quantum network. The main components of quantum networks and their roles are briefly discussed as follows.

Quantum nodes are computers equipped with quantum processor(s) and are capable of manipulations on qubits. Specifically, the nodes can establish quantum entanglements between their qubits and qubits in other nodes, or perform entanglement swapping among their own qubits in the case they work as repeaters.

Quantum entanglement and teleportation are the process of establishing a quantum link between two qubits on two nodes so that their states are interdependent on each other. In short, this process involves performing a Bell State Measurement [19] on qubits at two end-nodes then sending the co-relation through a classical transmission channel using two classical bits (e.g., via network slicing). Quantum entanglement is the means to start a process of sending data via a quantum network, called *teleportation* as shown in Fig. 1. Henceforth, we use the term "neighbors" to refer to nodes that are capable of forming *direct* entanglement with each other, i.e., they are connected through a classical channel, and their physical distance is set below 143km at this time[1].

Entanglement swapping is used to extend quantum entanglement to a long-distant pair of nodes. In this case, each end node can have a qubit entangled with a qubit in an *intermediate node* on the path connecting these two end nodes (i.e., repeater). The repeater then performs entanglement swapping on its own qubits which results in the entangled state between qubits of the two end nodes. We further refer to the intermediate node as a *repeater*. Since a repeater eventually needs to perform entanglement swapping, it must be equipped with at least two qubits.

Quantum channels refer to established entanglement pairs between two neighbor nodes. A chain of quantum channels that are connected, i.e., except for the source node and destination node, all repeaters are connected to exactly two channels in the chain, is referred to as a *quantum route*.

B. NETWORK MODEL: MATHEMATICAL NOTATIONS AND ILLUSTRATION

Mathematically, we model a quantum network as a graph $G = (V, E)$ in which $V = \{v_i\}_{i=1}^{N}$ represents the set of nodes in the network of size N, and $E = \{e_{i,j}; v_i, v_j \in V\}$ represents the set of *pairwise neighborhood relationships* among the nodes. More specifically, the neighborhood relationship between v_i and v_j exists when there is a possibility to form a quantum channel between them, in other words, qubits in v_i and v_j can establish quantum entanglements. To elaborate, there exists $e_{i,j} \in E$ only when v_i and v_j are connected through a classical network channel, and the physical distance between them is within the threshold to establish qubit entanglements (i.e. 143 km).

[1]A long-distance entanglement decays exponentially with the physical distance between the two entangled nodes [20]. Quantum teleportation over a distance of 143 km has been deployed but still in the early stage.

We define $C = \{c_i; c_i \in \mathbb{N}\}_{i=1}^{N}$ as the set of qubit capacities in which c_i is the *maximum* number of qubits that node v_i can generate to form entanglements with its neighbors. If there exists $e_{i,j} \in E$, v_i and v_j can establish at most $\min(c_i, c_j)$ entangled qubit pairs between them. In this paper, we assume that any nodes in the network can act as the source, destination, or repeater of a flow of traffic, therefore, we set the lower bound $\min(\{c_i;\ \forall\ c_i \in C\}) = 2$ so that any v_i can connect to at least two neighbors and perform entanglement swapping.

We then define a quantum route in the network between the source v_s and the destination v_d as a path $p = \{e_{p_1,p_2}, e_{p_2,p_3}, ..., e_{p_{k-1},p_k}\}$ with $e_{p_1,p_2}, e_{p_2,p_3}, ..., e_{p_{k-1},p_k} \in E$; $v_{p_1} = v_s$; $v_{p_k} = v_d$ and $p_i \neq p_j$ for any pair of (i, j) in the path; any nodes beside v_s and v_d are repeaters in the path. p is further constrained on the number of qubits available in the repeaters. Mathematically, let q_i be the number of qubits in a repeater v_i that has already been utilized in other quantum channels, the remaining number of available qubits must be at least two, i.e., $c_i - q_i \geq 2$. A demand (v_s, v_d) in the set of network demands $\mathcal{D}$ is successfully resolved by determining a quantum routing path between them that satisfies both mentioned constrains.

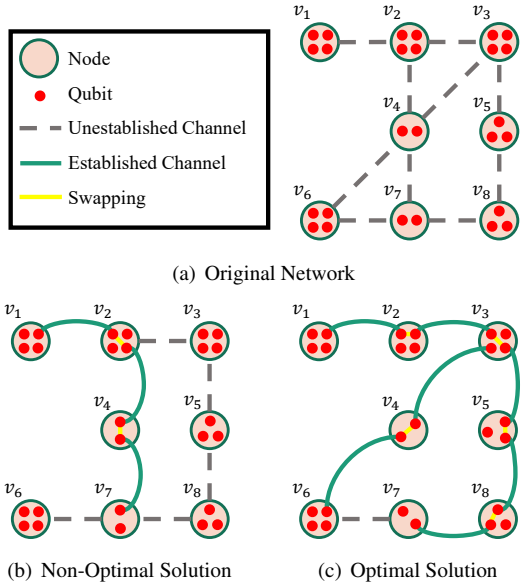
C. PROBLEM FORMULATION

In this work, we advocate a novel routing scheme – dubbed *Quantum Routing Scheme ($\mathcal{Q}_{\mathcal{RS}}$)* – that implements a deep reinforcement learning model to circumvent the computability limitations of *heuristic* conventional qubit assignment-based schemes, while concurrently meeting the quality-of-service requirements (e.g., connectivity) of networks. Given a quantum network $G = (V, E)$ with a set of qubit capacities and a set of network demands $\mathcal{D}$, our goal is to maximize the entanglement routing rate achieved by a routing scheme $\mathcal{Q}_{\mathcal{RS}}$ as follows:

$$\mathcal{Q}_{\mathcal{RS}} = \max_{(v_s,v_d)\in\mathcal{D}} \sum \mathcal{P}_{(v_s,v_d)} 1_{\{\mathcal{E}_{(p,q)}=1\ \forall\ (p,q)\in\mathcal{P}_{(v_s,v_d)}\}} \tag{1}$$

Where $\mathcal{P}_{(v_s,v_d)}$ denotes the entanglement path connecting the source-destination pair (v_s, v_d). Here, $1_{\{.\}}$ is the indicator function; it is equal to one if the condition in the subscript is true, otherwise zero. $\mathcal{E}_{(p,q)}$ denotes the entanglement status between the two quantum nodes p and q. We further constraint the size of $\mathcal{D}$ to be a fixed integer k. More specifically, $\mathcal{Q}_{\mathcal{RS}}$ performs routing on a window of k source-destination pair at a time. It should be noted that this constraint does not reduce the generality of the research problem, since a request set can be split or padded to meet the size requirement.

As the number of paths for a set $\mathcal{D}$ grows exponentially with $|V|$ and $|\mathcal{D}|$, determining the optimal solution for all requests simultaneously could become highly challenging. Furthermore, the network capacity may not be able to accommodate all requests in $\mathcal{D}$. Consequently, we break the problem into two tasks which are 1) to schedule demands to accommodate and 2) to determine the path for each one.



(a) Original Network

(b) Non-Optimal Solution

(c) Optimal Solution

FIGURE 2: An example on quantum network and routing solutions for two requests (v_1, v_7) and (v_3, v_6)

These two components make up the output of our $\mathcal{Q}_{\mathcal{RS}}$ scheme. We shows an illustrative example on how the two components affect the optimality of a solution in Fig. 2. Fig. 2(a) shows the original network topology with eight nodes and their maximum qubit capacities. At the same time, the network needs to accommodate two demands, (v_1, v_7) and (v_3, v_6). Fig. 2(b) demonstrates the case in which (v_1, v_7) is routed first using a least-hop strategy to select paths. In this case, (v_1, v_7) is assigned with the path $\{e_{1,2}, e_{2,4}, e_{2,7}\}$. This solution is not optimal as v_4 and v_7 do not have enough qubits left to accommodate the demand (v_3, v_6). On the other hand, Fig. 2(c) show the optimal case in which (v_3, v_6) is resolved first. Using the same least-hop strategy, both demands are sustained with two paths $\{e_{1,2}, e_{2,3}, e_{3,5}, e_{5,8}, e_{8,7}\}$ and $\{e_{3,4}, e_{4,6}\}$.

III. DEEP REINFORCEMENT LEARNING

Reinforcement learning [21] is a branch of deep learning in which an agent is trained to interact with an environment by observing its current states then taking actions. The training process is performed on a trial-and-error basis in which each action the agent takes in an environment state results in a return value, and the agent tries different actions based on the current states to eventually learn a mapping between states and actions that yields the highest returns. A Markov Decision Process (MDP) [21] is among the models used to represent a reinforcement learning problem. Mathematically, a MDP consists of the following components:

- A state space $\mathcal{S}_s$ that consists of all environment states
- An action space $\mathcal{A}_s$ that consists of all possible actions
- A reward function $\mathcal{R}(s^{(t)}, a^{(t)})$ with $s^{(t)} \in \mathcal{S}_s$, $a^{(t)} \in \mathcal{A}_s$, that maps a state-action combination $(s^{(t)}, a^{(t)})$ at step t to a scalar return value $R^{(t)}$
- A state transition function $\mathcal{T}$ that yields the probability of the agent taking an action $a^{(t)}$ at a state $s^{(t)}$ to move to the next state $s^{(t+1)}$: $\mathcal{T}(s^{(t)}, a^{(t)}, s^{(t+1)}) = p(s^{(t+1)}|s^{(t)}, a^{(t)})$

The reward function at a step t is computed as the sum of discounted future rewards:

$$\mathcal{R}(s^{(t)}, a^{(t)}) = \sum_{i=t}^{T} \lambda^{i-t} R(s^{(i)}, a^{(i)}), \tag{2}$$

where λ is a discount factor. A common algorithm to train a reinforcement learning agent is Q-Learning [22] in which a set of Q-values is utilized for the agent to select actions to take. The Q-values represents the return if the agent takes an action in a particular state, and an action is selected if it has the highest Q-value compared to others. Mathematically,

$$Q(s^{(t)}, a^{(t)}) = \mathcal{R}(s^{(t)}, a^{(t)}) + \lambda \sum_{s} \mathcal{T}(s^{(t)}, a^{(t)}, s^{(t+1)}) \max_{a^{(t+1)}} Q(s^{(t+1)}, a^{(t+1)}) \tag{3}$$

In terms of training, at the beginning, the agent randomizes a table of Q-values for all state/action combinations. After each action, the agent updates the corresponding Q-value using the following rule:

$$Q(s^{(t)}, a^{(t)}) \leftarrow (1-\alpha) Q(s^{(t)}, a^{(t)}) + \alpha(\mathcal{R}(s^{(t)}, a^{(t)}) + \lambda \max_{a^{(t+1)}} Q(s^{(t+1)}, a^{(t+1)}) - Q(s^{(t)}, a^{(t)})) \tag{4}$$

Recently, deep learning emerges with multiple breakthroughs in numerous areas [23]. The advantages of deep learning are threefold: 1) the deep neural network is able to *learn* to solve a problem with minimal guidance from the user; 2) the architecture of deep neural network allows for powerful representation capabilities in that the network can represent very complicated function; and 3) different designs of deep neural networks for different types of data can be aggregated in one model which further increases representation powers. Reinforcement learning also gets numerous benefits from the advancement of deep learning. Various deep reinforcement schemes have been proposed with different applications with great successes [24]. Particularly in resource management in which the goal is to optimize the usages of available resources over a set of tasks, deep reinforcement learning (DRL) has been showed to solve challenges of heuristic methods including 1) the impossibility to model a complex system accurately, 2) noisy inputs and diverse operating conditions may affect the decision making of algorithms in practice, and 3) certain performance metrics are difficult to optimized [25]. In resource management, DRL is widely applied in domain such as communication and networking [26], Internet of Things [27], 5G networks [28], for tasks such as scheduling, routing, rate control, security, quality of services, etc. In quantum networking, the only other work that utilizes DRL [17] focuses on optimizing entanglement times on quantum channels across a path to ensure an entanglement state between the two end nodes can be established before any channels decay. In this paper, we

instead aim to optimize the allocation of qubits in a network to maximize the number of accommodated communication requests.

In general, deep reinforcement learning uses a deep neural network to represent the mapping between input states to actions and rewards. Let the function of a deep neural network be $D(\cdot)$ then

$$(a^{(t)}, R^{(t)}) = D(s^{(t)}) \tag{5}$$

In this paper, we utilize two deep reinforcement learning approaches. The first uses a deep neural network to represent the reward function $\mathcal{R}(\cdot)$. In this case, we train the reward network using an explicit reward function that is designed towards the optimization goal in Eq. (1). In the second case, we utilize a Deep Q Network (DQN) [18]. In DQN, the neural network is trained to predict the Q value of the action. Two version of the deep neural networks are maintained during training, a predict network and a target network. The predict network takes the input state at t, generates $Q^{(t)}$ values for the selected actions. The input state is then updated to $t+1$ and fed to the target network to generates the $Q^{(t+1)}$ values at $t+1$. The network is trained so that $Q^{(t)}$ converges to $Q^{(t+1)}$. The predict network and target network share the same weights at the beginning. However, the predict network is updated constantly during training, whereas the target network is updated with the weights from the predict network once every few hundred epochs.

IV. DEEP QUANTUM ROUTING AGENT

In this section, we describe our deep reinforcement routing scheme to which henceforth referred as Deep Quantum Routing Agent (DQRA). DQRA utilizes a novel deep neural network to schedule requests and a shortest path algorithm for routing them. Depending on the training algorithm, the deep neural networks is either a deep reward network (DRN) or a deep Q network (DQN). Since the network architectures are identical in both cases, for simplicity, we call the neural network a schedule neural network (SNN) throughout this section. Given a quantum network $G(V, E)$ with a qubit capacity set C and a set of routing requests $\mathcal{D}$, we define the accommodation of all requests in $\mathcal{D}$ an *episode*, denoted as $\mathcal{E}$. $\mathcal{E}$ consists of a chain of consecutive action $a^{(t)}$ that DQRA makes at each step t to resolve *one* request in $\mathcal{D}$.

Mathematically, at step t in $\mathcal{E}$, the input of DQRA includes the current network topology $G^{(t)}(V, E^{(t)})$, the current qubit capacity $C^{(t)}$, and the current request set $\mathcal{D}^{(t)}$. DQRA then makes an action $a^{(t)}$ which represents the selection of source-destination pair to accommodate at step t. After that, a shortest path algorithm is utilized to determine the path for the selected request. DQRA decides the actions in an episode using a reward function that is designed to guide the model to solutions that fulfills more requests in an episode. An episode ends when all requests in $\mathcal{D}$ are resolved, or when no current requests can be accommodated. We demonstrate the input and output of DQRA in Fig. 3.

In the following subsections, we discuss in details our deep reinforcement routing agent for quantum networks in terms of input states, output actions, reward function, model architecture, and training.

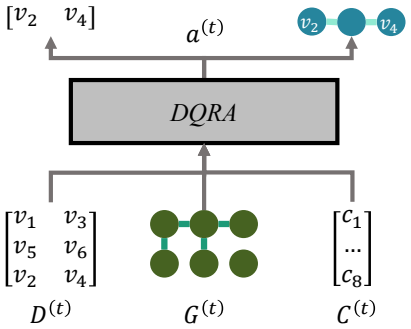


FIGURE 3: An example of input and output of DQRA at each step t

A. INPUT STATE

At step t in an episode $\mathcal{E}$, an input state to DQRA consists of 1) the connectivity in the network $G^{(t)}(V, E^{(t)})$, 2) the qubit capacity of each node $C^{(t)}$, and 3) the request set $\mathcal{D}^{(t)}$. All three inputs are fed to the SNN component of DQRA to decide which request to resolve, then $G^{(t)}(V, E^{(t)})$ and $C^{(t)}$ are utilized by the shortest path component to route the selected request. First, we discuss the data representation for each input type.

In terms of network topology, $G^{(t)}$ is generated as a subset of G in which edges that connect to nodes with current qubit capacity of 0 are removed:

$$\begin{aligned} G^{(t)} &= (V, E^{(t)}); \\ E^{(t)} &= \{e_{i,j}; e_{i,j} \in E; C_i^{(t)} > 0; C_j^{(t)} > 0\} \end{aligned} \tag{6}$$

Then, we use a binary row vector of which elements correspond to edges that originally exist in G. Let the input vector that represents $G^{(t)}$ be $\mathcal{A}^{(t)}$, and $\mathcal{A}_k^{(t)}$ be the element that corresponds to edge $e_{i,j} \in G$, then

$$\mathcal{A}_k^{(t)} = \begin{cases} 1 & e_{i,j} \in G^{(t)} \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

At each step t in an episode, the current request set $\mathcal{D}^{(t)}$ is another input to DQRA. We aim for the deep network to analyze all pending requests before making decision with this design of inputs. In terms of representation, each node in a request is modeled as binary vector $v = [\{v_i; i = 1...|V|\}]$ in which

$$v_i = \begin{cases} 1 & \text{if current node is } i \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

If a request is already resolved prior to step t, its source and destination are replaced with 0-vectors. Overall, each source-destination pair makes up one row in $\mathcal{D}^{(t)}$ which in

turns become a binary matrix of size $k \times 2|V|$.

$$\mathcal{D}^{(t)} = \begin{bmatrix} v_{s_1} & v_{d_1} \\ v_{s_2} & v_{d_2} \\ \dots & \dots \\ v_{s_{|\mathcal{D}|}} & v_{d_{|\mathcal{D}|}} \end{bmatrix} \tag{9}$$

Finally, the qubit capacity at step t is modeled as a row vector $\mathcal{C}^{(t)} = [\{c_i^{(t)}; c_i \in C\}]$. All three components make up the state $X^{(t)}$ that DRN observes from the environment at step t:

$$X^{(t)} = \{\mathcal{A}^{(t)} \quad \mathcal{D}^{(t)} \quad \mathcal{C}^{(t)}\} \tag{10}$$

Overall, the algorithm takes $X^{(t)}$ as input and outputs an action that is described in the next subsection.

B. OUTPUT ACTION

In short, DQRA outputs actions that represent the routing schedule for requests in $\mathcal{D}$ and their paths. More specifically, at each step t in an episode, DQRA selects a single request in the set of pending requests $\mathcal{D}^{(t)}$ then assigns a path connecting the source and destination nodes. In general, a deep neural network can be designed to output paths in a graph, for example, as a binary vector of which elements represent whether an edge is presented in the path or not. However, the number of edges in a graph may be exceedingly high, which, when coupled with the already high dimensional input, may lead to a highly complex neural network that is not ideal for real time settings. Accordingly, we utilize a hybrid approach, that is to use the deep network SNN to output the schedule of routing requests, and a shortest path algorithm with customized metric to determine the path for each request. Both components, however, contribute to the computation of rewards for the actions, and therefore both impact how the deep network is trained.

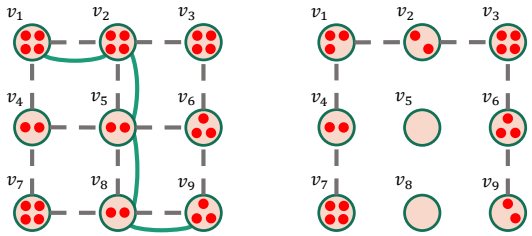
For scheduling, SNN outputs a reward vector (or a Q vector the case of DQN) $r^{(t)}$ of size $|\mathcal{D}|$: $r^{(t)} = \{r_0^{(t)}, r_1^{(t)}, \dots, r_{|\mathcal{D}|}^{(t)}\}$ in which $r_i^{(t)}$ represents the reward at the end of step t if DQNA routes request i. The request to be selected is the one with the highest reward

$$a^{(t)} = \text{argmax}_i(r_i^{(t)}) \tag{11}$$

With $a^{(t)}$ selected, to determine the path between the source and destination, we utilize a shortest path approach in which the metric expresses the qubit capacity of the nodes. More specifically, a shortest path algorithm is applied in which the weight $w_{i,j}^{(t)}$ of an edge $e_{i,j}$ at step t in the episode is as

$$w_{i,j}^{(t)} = \frac{1}{\min(c_i^{(t)}, c_j^{(t)})} \tag{12}$$

where $c_i^{(t)}$ is the current capacity of node v_i. It can be seen that $w_{i,j}^{(t)}$ is high when either nodes connected to $e_{i,j}$ has low capacity and low otherwise. Therefore, a shortest path of which total metric is low is more likely to traverse nodes with high qubit capacities and less likely to exhaust qubits in any nodes. Overall, this path selection approach focuses on preserving qubit capacities of nodes in each step, which in turn extends the number of requests that can be accommodated in an episode. To prevent a node with less than two qubits being selected as a repeater, we apply the shortest path algorithm on a node-induced sub-graph $\mathcal{G}^{(t)}$ of $G^{(t)}$ in which a node v_i is retained only if it has $c_i^{(t)} \geq 2$ *or* v_i is either the source or destination in the current request.



(a) $\mathcal{G}^{(t)}$ (b) $\mathcal{G}^{(t+1)}$

FIGURE 4: An example on updating state $G^{(t)}$ to $G^{(t+1)}$ after selecting and routing request (v_1, v_9)

Since the size of the output reward vector is constant at $|\mathcal{D}|$, it is possible that a node v_i is repeatedly selected throughout an episode. To address this issue, we maintain a list L_r of requests that are already resolved. If the the highest reward belongs to a request that is already in L_r, the network moves on to the next highest reward request. In short, the *pending* request with the highest reward value is selected in each step.

At the end of each action, the input states are updated for the next step $t+1$. First, the row that associates to the request resolved by $a^{(t)}$ in $\mathcal{D}^{(t)}$ is replaced with 0's to obtain $\mathcal{D}^{(t+1)}$. Then, $\mathcal{C}^{(t)}$ is updated with new qubit capacities. Particularly, given a routing path $\mathcal{P}^{(t)} = \{e_{s^{(t)},p_1}, e_{p_1,p_2}, \dots, e_{p_k,d^{(t)}}\}$ assigned to the selected request $(v_{s^{(t)}}, v_{d^{(t)}})$, each node v_i capacity is updated as

$$c_i^{(t+1)} \leftarrow \begin{cases} c_i^{(t)} - 1 & \text{if } i \in \{s^{(t)}, d^{(t)}\} \\ c_i^{(t)} - 2 & \text{if } i \in \{p_1, p_2, ..., p_k\} \\ c_i^{(t)} & \text{otherwise} \end{cases} \tag{13}$$

Finally, $G^{(t+1)}$ is acquired by removing all edges that associate with any node v_i that has $c_i^{(t+1)} = 0$ and updating the weights of the edges using $\mathcal{C}^{(t+1)}$ and Eq. (12). We illustrate the updating from input state $G^{(t)}$ to $G^{(t+1)}$ in Fig. 4. At $\mathcal{G}^{(t)}$, the request to accommodate is (v_1, v_9); the routing path is $\{e_{1,2}, e_{2,5}, e_{5,8}, e_{8,9}\}$. After assigning the path, $G^{(t+1)}$ has the same nodes with new capacities, and v_5 and v_8 are disconnected from the rest of the network since they do not have any available qubits left.

C. REWARD FUNCTION

As our research problem is to maximize the number of accommodated routing requests in a window, we specifically design a reward function that addresses this target. In details, we directly associate the rewards that the model receives with the current and future numbers of resolved requests in an

episode. Mathematically, let the reward at step t be $R^{(t)}$, then

$$R^{(t)} = n_r^{(t)}\alpha + (|\mathcal{D}| - n_r^{(t)})\beta f + \lambda R^{(t+1)}(1-f) \quad (14)$$

where $n_r^{(t)}$ is the number of requests that are resolved after step t, f is a binary indicator that is 1 when t is the ending step of and episode and 0 otherwise, $\alpha > 0$ is the reward term, $\beta < 0$ is the penalty term, and $\lambda \in [0, 1]$ is a discount factor. Both α, β, and λ are hyper-parameters to be selected during the training phase. Since the computation of $R^{(t)}$ requires future reward values, we let DQRA finish an episode to obtain the final reward value, then gradually trace back to calculate the rewards of prior steps. We select α, β, and λ so that the reward for one action is significantly negative if it leads to a failed episode (i.e., ending without being able to solve all requests) in the next few steps. On the other hand, actions that lead to a successful episode should be rewarded with positive value. Overall, the reward function design encourages the agent to find schedules that accommodate more requests, and penalizes schedules that end too early - the less requests solved, the heavier the penalties.

D. DEEP REWARD NETWORK ARCHITECTURE

In this subsection, we describe the architecture of the deep reward network, DRN, in DQRA. As described in subsection §IV-A, the input to the deep reward network at step t in an episode consists of three components 1) $\mathcal{D}^{(t)}$ that represents the set of pending requests, 2) $\mathcal{A}^{(t)}$ that encodes the network graph state, and 3) $\mathcal{C}^{(t)}$ that expresses the qubit capacities of the nodes. We make two observations on the input states. First, all components may have high dimensionality when the graph size increases. Second, $\mathcal{D}^{(t)}$ is very sparse, and $\mathcal{A}^{(t)}$ may also become relatively sparse after the first few steps. Overall, feeding all components to a same layer is not ideal. Consequently, we first feed each component into a different embedding network. In short, an embedding network consists of multiple fully-connected layers which transform an input vector into embedding vectors, usually of lower dimensionality. Let the mappings represented by the embedding networks for $\mathcal{D}^{(t)}$, $\mathcal{A}^{(t)}$, and $\mathcal{C}^{(t)}$ be $M_{\mathcal{D}}(\cdot)$, $M_{\mathcal{A}}(\cdot)$, and $M_{\mathcal{C}}(\cdot)$, respectively, then

$$\begin{aligned} M_{\mathcal{D}}(\mathcal{D}^{(t)}) = U_{\mathcal{D}} &= \begin{bmatrix} u_{s_1,d_1} \\ u_{s_2,d_2} \\ \cdots \\ u_{s_{|\mathcal{D}|},d_{|\mathcal{D}|}} \end{bmatrix} \\ M_{\mathcal{A}}(\mathcal{A}^{(t)}) &= u_{\mathcal{A}^{(t)}} \\ M_{\mathcal{C}}(\mathcal{C}^{(t)}) &= u_{\mathcal{C}^{(t)}} \end{aligned} \quad (15)$$

It can be seen that the output of $M_{\mathcal{D}}(\cdot)$ is a matrix of $|\mathcal{D}|$ vectors whereas that of $M_{\mathcal{A}}(\cdot)$ and $M_{\mathcal{C}}(\cdot)$ are two single row vectors. The reason for this design is that we want the reward network to further learn the interrelationships among all the pending requests using a *Self-Attention* layer [29], [30]. In short, an attention layer takes input as three matrix Q, K, and V, and output a "context" matrix $\mathcal{S}$ as follows

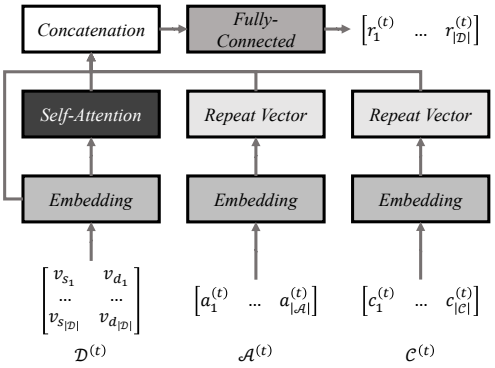


FIGURE 5: DQRA deep neural network architecture

$$\mathcal{S} = \delta(\frac{QK^T}{\sqrt{|K|}})V \quad (16)$$

where δ is the SoftMax function, and K^T is the transposed matrix of K. In $\mathcal{S}$, each row represents the "context score" of the corresponding row in Q with respect to all rows in K and V. In a self-attention layer, Q, K, and V are identical, and in our context, they are equal to $U_{\mathcal{D}}$. Overall, our purpose of using a self-attention layer is to obtain a score matrix that represents the "context" of each pending request with respect to the rest. Let the context matrix $S_{\mathcal{D}}$ and the mapping by the self-attention layer be $S(\cdot)$, then

$$\mathcal{S}_{\mathcal{D}} = S(U_{\mathcal{D}}) = \begin{bmatrix} \sigma_{s_1,d_1} \\ \sigma_{s_2,d_2} \\ \cdots \\ \sigma_{s_{|\mathcal{D}|},d_{|\mathcal{D}|}} \end{bmatrix} \quad (17)$$

We then repeat $u_{\mathcal{A}^{(t)}}$ and $u_{\mathcal{C}^{(t)}}$ $|\mathcal{D}|$ times, then concatenate them with the rows in $U_{\mathcal{D}}$ and $\mathcal{S}_{\mathcal{D}}$ to form the complete embedding for each request:

$$U = \begin{bmatrix} u_{s_1,d_1} & \sigma_{s_1,d_1} & u_{\mathcal{A}^{(t)}} & u_{\mathcal{C}^{(t)}} \\ u_{s_2,d_2} & \sigma_{s_2,d_2} & u_{\mathcal{A}^{(t)}} & u_{\mathcal{C}^{(t)}} \\ \cdots & & & \\ u_{s_{|\mathcal{D}|},d_{|\mathcal{D}|}} & \sigma_{s_{|\mathcal{D}|},d_{|\mathcal{D}|}} & u_{\mathcal{A}^{(t)}} & u_{\mathcal{C}^{(t)}} \end{bmatrix} \quad (18)$$

Finally, U is input into a block of fully-connected layers that output a vector of $|\mathcal{D}|$ values $r_1^{(t)}, r_2^{(t)}, \dots, r_{|\mathcal{D}|}^{(t)}$ that represent the reward if DQRA select each request. The architecture of DQRA's deep neural network is shown in Fig. 5.

E. TRAINING ALGORITHM

We utilize two algorithms to train DQRA deep neural network, specifically, as a supervised deep reward network (DRN), and as a deep Q network (DQN). In the supervised case, DRN is trained as a supervised model to predict the true reward of taking an action. We generate the training data as follows. At the beginning of an episode, we randomize the request set $\mathcal{D}$. For each step in an episode, the three input components are fed to the model to generate actions. The inputs are then updated for the next step based on the action DQRA makes using the process that is described in Section

§IV-B. This action-update process repeats until the episode ends by either fulfilling all requests or by DQRA not being able to accommodate any more pending requests. At the end of an episode, the reward values for each step are computed backward using equation (14). Overall, for an episode $\mathcal{E}_i$, the training data (in the format [features] ; [label]) obtained is as

$$\mathcal{E}_i = \begin{bmatrix} \mathcal{A}^{(0)} & \mathcal{D}^{(0)} & \mathcal{C}^{(0)} \\ \mathcal{A}^{(1)} & \mathcal{D}^{(1)} & \mathcal{C}^{(1)} \\ \dots & \dots & \dots \\ \mathcal{A}^{(T)} & \mathcal{D}^{(T)} & \mathcal{C}^{(T)} \end{bmatrix} ; \begin{bmatrix} R^{(0)} \\ R^{(1)} \\ \dots \\ R^{(T)} \end{bmatrix} \tag{19}$$

with T being the number of steps in $\mathcal{E}_i$. Following the Experience Replay [31] method, data from each episode are stored and resampled for future training iterations. In terms of training objective, we utilize *Mean Squared Error* function. However, it can be seen that, at each step t, the output of the reward network is a vector of size $|\mathcal{D}|$ while the target $R^{(t)}$ is a scalar. This is because the model only knows the reward of actions that it has taken. To address this problem, we utilize a *mask* vector $m^{(t)}$ of size $|\mathcal{D}|$ in which $m_i^{(t)} = 1$ if request i is selected in $a^{(t)}$, and 0 otherwise. The loss for one episode $\mathcal{E}_i$ is computed as

$$\mathcal{L}_i = \sum_{t \in \text{episode}} (r^{(t)} * m^{(t)} - R^{(t)} m^{(t)})^2 \tag{20}$$

where $*$ represents an element-wise multiplication, i.e., if x and y are two vectors of size n, $x * y = [x_1 y_1 \quad \dots \quad x_n y_n]$. By using the mask vector, we prevent rewards from unselected requests to participate in the loss function. Finally, the overall loss is averaged across episodes in a training batch of size n_b

$$\mathcal{L} = \frac{1}{n_b} \sum_{i \in \text{batch}} \mathcal{L}_i \tag{21}$$

$\mathcal{L}$ can be used to train the reward network with any neural network training algorithms such as Stochastic Gradient Descent (SGD) [32] or ADAM [33].

In the DQN training scheme, the output of DRN is considered the Q values of each action, and the loss of one episode $\mathcal{E}_i$ is as

$$\mathcal{L}_i = \sum_{t \in \text{episode}} (Q_P^{(t)} * m^{(t)} - Q_P'^{(t)} * m^{(t)})^2 \tag{22}$$

where $Q_P^{(t)}$ is the output of the predict network at step t, and $Q_P'^{(t)}$ is the target Q value at step t which is computed as follows

$$Q_P'^{(t)} = (1 - l_r) * Q_P^{(t)} + l_r * (Q_T^{(t+1)} + \lambda R^{(t)}) \tag{23}$$

where l_r is a learning rate, $Q_T^{(t+1)}$ is the Q values output by the target network for $t+1$, λ and $R^{(t)}$ are the discount factor and reward function at t as described in Eq. (14). For DQN, we still utilize the mask vector $m^{(t)}$ to remove rewards of unselected actions from the loss function. The overall loss for one training batch is also calculated by Eq. (21). The predict network can be trained using regular algorithms like SGD or ADAM; the target network is updated with the predict network's weights once every few hundred of epochs. During decision making phases, only the predict network is utilized.

V. EXPERIMENT STUDY

All experiments are implemented in Python 3.6. We also want to acknowledge the use of the Python packages Numpy [34], Matplotlib [35], NetworkX [36], and Tensorflow [37]. All experiments are performed on a workstation with Intel(R) Core(TM) i9-9940X CPU, 128GB of RAM, and four Nvidia Quadro P6000 graphic cards.

In this experiment study, we focus on grid networks of size $n_G \times n_G$ nodes, and use a routing window size of n_G requests, with $n_G \in \{5, 6, 7, 8, 9, 10\}$. We test four cases with different nodes' qubit capacities: 1) $c_i = 2 \ \forall \ i$; 2) $c_i \in [2, 4] \ \forall \ i$; 3) $c_i \in [3, 4] \ \forall \ i$; and 4) $c_i = 4 \ \forall \ i$. The requests in each window are completely randomized in terms of space, i.e., sources and destinations. Furthermore, in the current stage of this research, we are not considering the time component of entanglement. In other words, our assumption is that all entanglements are maintained throughout their routing windows. The test cases represent networks with increasingly more limited qubit capacity. In all cases, a node can act as repeater twice within one window only if it has the maximum number of qubits of 4 at the beginning of an episode. The simulation with $c_i = 2 \ \forall \ i$ represents the extreme case in which each node can act as a repeater for exactly one path. Furthermore, in this case, if a node is the source or destination of a request, it cannot be a repeater anymore. In all cases, each node can be selected as the source or destination of requests within a window no more than once. Finally, in all experiments, it is possible that the network does not have enough resources to accommodate all requests in a window, therefore routing solutions at 100% request-accommodation rates are not to be always expected.

In terms of modeling, we utilize the same architectures (i.e., equal numbers of layers and neurons) of deep networks in both cases for both the DQRA using supervised reward network (denoted as DQRA-DRN) and DQRA using the deep Q network (DQRA-DQN) to ensure fair comparisons. To simplify the hyper-parameter space, we set the number of layers in the three embedding components to be the same; each layer has an equal number of neurons to its input dimensionality and focus on fine-tuning the number of layers for each component (2, 3, and 4). After fine-tuning, the selected deep network architectures are as in Tab. 1. The architecture of each component is denoted as $(l_1, l_2, \dots)$ in which l_1 is the number of neurons in the first layer, l_2 the second layer, and so on. In short, the three embedding blocks have two layers, and the decision making block (that outputs returns for actions) has three layers of $6|\mathcal{D}|$ neurons. The hyper-parameters α, β, and λ in Eq. (14) are set at 0.2, -1, and 0.9, respectively. The learning rate l_r for training DQRA-DQN as in Eq. (14) is 0.1. Finally, the mini-batch size in all experiments, n_b in Eq. (21), is 512.

We implement two baseline routing strategies to compare with DQRA. The first (Random) performs random selection

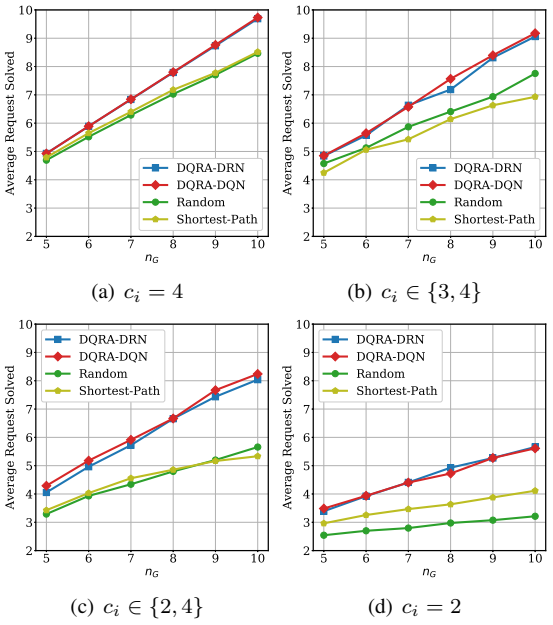
TABLE 1: Schedule neural network architecture by n_G

n_G	$M_{\mathcal{D}}$	$M_{\mathcal{A}}$	$M_{\mathcal{C}}$	*F. Connected*	*No. Para.*
5	(50,50)	(80,80)	(25,25)	(150,150,150)	95,711
6	(72,72)	(120,120)	(36,36)	(216,216,216)	201,193
7	(98,98)	(168,168)	(49,49)	(294,294,294)	376,559
8	(128,128)	(224,224)	(64,64)	(384,384,384)	647,489
9	(162,162)	(288,288)	(81,81)	(486,486,486)	1,043,695
10	(200,200)	(360,360)	(100,100)	(600,600,600)	1,598,921

on requests, then assigning path to the selected request using the shortest path algorithm. The second (Shortest Path) strategy selects requests based on a shortest-first strategy. Specifically, in each step of an episode, the shortest paths for each pending requests are first determined (independently to each other), then the one with minimum value is chosen to accommodate.

We use the average number of successfully solved requests across 1000 windows as the evaluation metric. The five-run averaged experiment results are shown in Fig. 6. In all experiments, the deep networks are trained for $10,000$ epochs. As can be seen from Fig. 6, the two versions of DQRA yield similar performances in all tests, both of which are significantly higher than the two naive baselines in all test cases. In Fig. 6(b) and 6(c) where each node has c_i randomized, DQRA models maintain a successful routing rates of over 80% in all network sizes, with DQRA-DQN being slightly better than DQRA-DRN. In the two extreme cases where each node has exactly four and two qubits (Fig. 6(a) and (d)), the performances of DQRA models are almost identical. In networks with $c_i = 4 \ \forall \ i$, the two achieve almost 100% resolving rates, whereas in networks with $c_i = 2 \ \forall \ i$, they achieve the rates from about 70% at $n_G = 5$ to approximately 59% at $n_G = 10$.

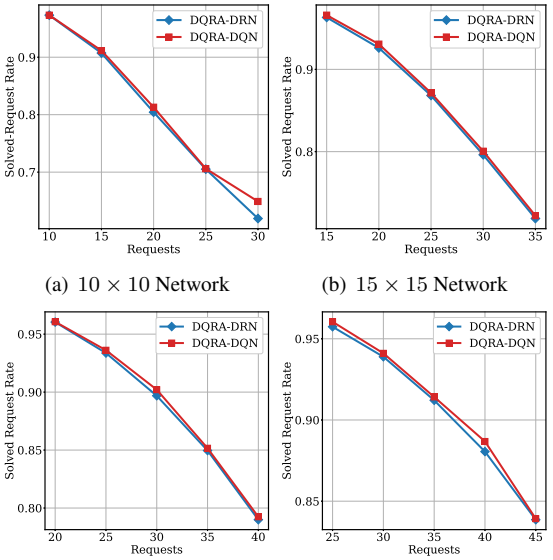
Next, we analyze the performances of the two DQRA models in high scale settings. In this experiment, the networks are of size $n_G \times n_G$ nodes with $n_G \in \{10, 15, 20, 25\}$ and increasing window sizes $|\mathcal{D}| \in \{n_G, n_G + 5, n_G + 10, n_G + 15, n_G + 20\}$. The nodes in all experiments have a fixed capacity of four qubits. We no longer consider Random model and Shortest Path model as they have been shown to be inferior previously. The result of this experiment is shown in Fig. 7. The evaluation metric that is used is the rate of fulfilled requests across 5,000 windows. All models are trained for 10,000 epochs. It can be seen that in all network sizes, the two DQRA models performs almost identically with DQN always performing slightly better. Furthermore, as the network size increases, the solved request rates of the model drop more slowly. All models' solving rates start around 95% at $|\mathcal{D}| = n_G$, however, at $|\mathcal{D}| = n_G + 20$, the two models solve less around 65% requests in 10×10 networks, 70% in 15×15 networks, 80% in 20×20 networks, and around 85% in 25×25 networks. This result is to be expected, since larger network sizes yield more alternative routes and therefore the network can accommodates more requests.



(a) $c_i = 4$ (b) $c_i \in \{3, 4\}$

(c) $c_i \in \{2, 4\}$ (d) $c_i = 2$

FIGURE 6: Models' performance on routing n_G requests in grid networks of size $n_G \times n_G$ nodes of qubit capacity c_i

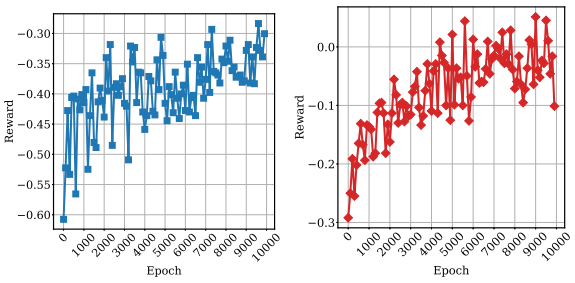


(a) 10×10 Network (b) 15×15 Network

(c) 20×20 Network (d) 25×25 Network

FIGURE 7: DQRA models' performances in high scale (i.e. large numbers of nodes and requests) settings

We further investigate the learning capabilities of the deep models. Figure 8 illustrates the rewards value of DRN and DQN throughout 10,000 training epochs in a network of 10×10 nodes with a window of 10 requests. Reward values are computed after every 100 epochs by averaging reward from 1,000 requests. Since the requests are randomized, we observe the fluctuations as can bee seen in the figures. Nevertheless, we see that training rewards in both models effectively increase during around the first 8,000 epochs. After that, DQN seems to stop learning while DRN continues to learn albeit considerably more slowly.



(a) DRN (b) DQN

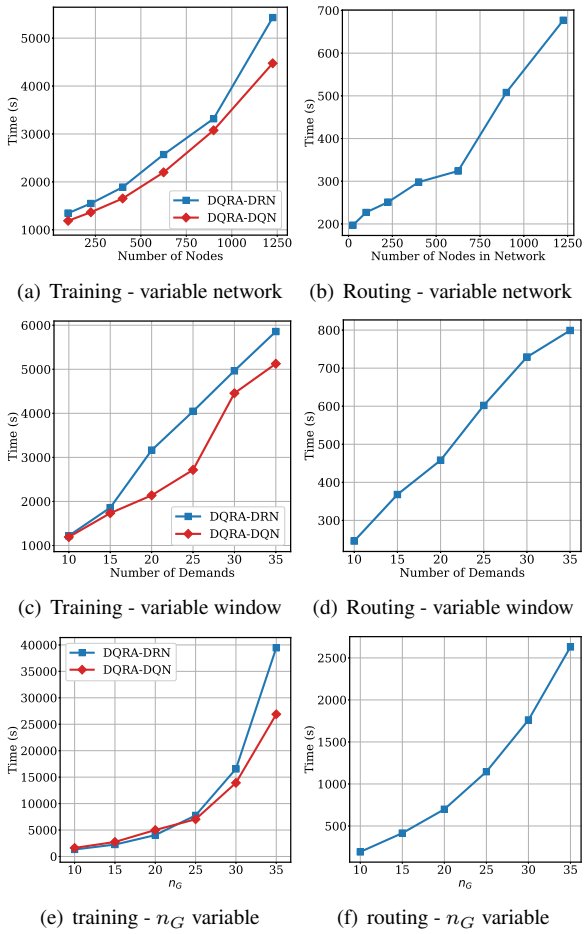
FIGURE 8: Training reward of DRN and DQN in a 10×10 network with window of 10 in 10,000 epochs

Finally, we examine the training time and routing time of DQRA models. Since the deep architectures used in both versions of DQRA are identical, and DQRA-DQN only uses the predict network when making decision, the two DQRA models should have equal routing time performances. Their training times, however, vary due to having different training algorithms. In this experiment, we simulate three cases:

1) Constant window size of 10, network sizes $n_G \times n_G$ with $n_G \in \{10, 15, 20, 25, 30, 35\}$. This effectively means the numbers of nodes in the network being $\{100, 225, 400, 625, 900, 1225\}$, respectively.
2) Constant network size of 10×10 nodes, window sizes increase with $|\mathcal{D}| \in \{10, 15, 20, 25, 30, 35\}$.
3) Network sizes of $n_G \times n_G$, and window size of n_G with $n_G \in \{10, 15, 20, 25, 30, 35\}$. In other words, both network sizes and window sizes increase in this experiment.

To ensure an equal number of routing between the two models, we set the qubit capacity of all nodes to a very high number so all requests of the windows are always accommodated. In all cases, training times are measured for 1,000 training epochs, and routing times are measured for 1,000 windows. As can be seen in Fig. 9(a)(c)(d), training times of DQRA-DQN are slightly lower than DQRA-DRN in all three experiments. We further observe that, training time increases as a polynomial function of network size (Fig. 9(a)) and a linear function of window size (Fig. 9(c)). In Fig. 9(e) where both network size and window size increase, training times of both models increase more sharply, however, still with polynomial patterns. On the other hand, within the observed experimental space, routing times of the models increase with a linear pattern when network size and window size increase individually, and turn into polynomial pattern when both increase simultaneously.

We deem the result of the experiment on time performance to be expected. In general, the complexity of neural networks, which majorly involve matrix multiplications and summations, is a polynomial function of their number of parameters. In this case, this number is a quadratic function of the network size due to the use of the adjacency matrix in the input; and our designs make the number of parameters in a network independent on the demand window sizes.



(a) Training - variable network (b) Routing - variable network

(c) Training - variable window (d) Routing - variable window

(e) training - n_G variable (f) routing - n_G variable

FIGURE 9: Models' training times and routing times when network sizes increases (a)(b), window size increases (c)(d), and both network and window sizes increase (e)(f)

Furthermore, the shortest path algorithm is utilized exactly once for any requests in a window. A final note to be emphasized is that, while having lower training times, DQRA-DQN must maintain two deep neural networks (a prediction network and a target network) during training, and therefore is more computationally intensive in this case. Therefore, it could be challenging to train DQRA-DQN for a large scale network in a system with limited computational resources. We show the number of parameters in the scheduling neural network in quantum networks of size $n_G \times n_G$ with $n_G \in \{5, 10, 15, 20, 25, 30, 35\}$ in Fig. 10. It can be seen that the schedule neural network have to maintain approximately 250 million parameters when the quantum network size grows to 1225 nodes.

VI. CONCLUSION

Routing in quantum networks, one of the key problems of the next-generation network system, and machine learning, one of today leading research areas, have not seen much integration currently. With such motivation, this paper aims to bridge that gap with a new machine-learning-powered quantum routing model for quantum networks. In particular, given a quantum network and a set of communication

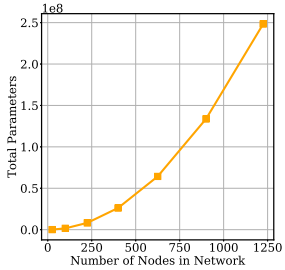


FIGURE 10: Total number of parameters in schedule neural network in quantum networks of size $n_G \times n_G$

demands, we have proposed a deep reinforcement routing scheme (DQRA) to construct routing paths for all demands in the network.

We have modeled the problem of entanglement routing in quantum networks as a reinforcement learning problem that consists of input states, actions, and rewards. At each step in a routing window, we construct input states of the model by considering the quantum network's states, the qubit capacities of nodes, and the set of pending demands; the actions were defined as the demand selection to accommodate and the path connecting the two ends of the demand; and the rewards were designed to guide the model towards schedules that fulfill the maximum number of demands in a time window. DQRA consists of two components, an empirically designed deep neural network that was used to observe the current input states to decide the routing schedule and routing paths of the selected demands were then determined by a qubit-preserved shortest path algorithm. We further utilized two algorithms to train the deep network of DQRA, as a deep reward network, and as a deep Q network (DQN). Our experiment study shows that DQRA is able to maintain a rate of successfully routed requests at above 80% on average in a qubit-limited network and approximate 60% in extreme conditions (i.e. each node in the network can be repeater exactly once in a window). We also empirically shown that DQRA is scalable with topology sizes and window sizes.

This work will lay foundation for the research on using applied machine learning to leverage quantum routing in quantum networks. For future works, we will examine the following aspects: 1) incorporating the success rates of qubit entanglement and qubit swapping into the input states of the models, 2) exploring other designs of the reward functions so that our model can route requests on other basis such as fidelity or entanglement rates, and 3) investigating different deep architecture designs to maximize the agents' performances while further reducing their complexities.

ACKNOWLEDGEMENT

This research was supported in part by the US NSF grant CNS-2103405.

REFERENCES

[1] S. Shi and C. Qian, "Concurrent entanglement routing for quantum networks: Model and designs," ser. SIGCOMM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 62–75.

[2] C. Cicconetti, M. Conti, and A. Passarella, "Request scheduling in quantum networks," IEEE Transactions on Quantum Engineering, vol. 2, pp. 2–17, 2021.

[3] W. Dai, T. Peng, and M. Z. Win, "Optimal remote entanglement distribution," IEEE Journal on Selected Areas in Communications, vol. 38, no. 3, pp. 540–556, 2020.

[4] C. Li, T. Li, Y.-X. Liu, and P. Cappellaro, "Effective routing design for remote entanglement generation on quantum networks," npj Quantum Information, vol. 7, pp. 1–12, 2020.

[5] A. K. Arute, F. and R. e. a. Babbush, "Quantum supremacy using a programmable superconducting processor," Nature, vol. 574, 2019.

[6] A. Harrow and A. Montanaro, "Quantum computational supremacy," Nature, vol. 549, 2017.

[7] L. Gyongyosi and S. Imre, "Scalable distributed gate-model quantum computers," Nature, vol. 11, 2021.

[8] S. Aaronson and L. Chen, "Complexity-theoretic foundations of quantum supremacy experiments," in Proceedings of the 32nd Computational Complexity Conference, ser. CCC '17. Dagstuhl, DEU: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017.

[9] J. Chung, G. S. Kanter, N. Lauk, R. Valivarthi, W. Wu, R. R. Ceballos, C. Pena, N. Sinclair, J. Thomas, S. Xie, R. Kettimuthu, P. Kumar, P. Spentzouris, and M. Spiropulu, "Illinois express quantum network (ieqnet): metropolitan-scale experimental quantum networking over deployed optical fiber," in Defense + Commercial Sensing, 2021.

[10] A. Dahlberg, M. Skrzypczyk, T. Coopmans, L. Wubben, F. Rozpundefineddek, M. Pompili, A. Stolk, P. Pawełczak, R. Knegjens, J. de Oliveira Filho, R. Hanson, and S. Wehner, "A link layer protocol for quantum networks," in Proceedings of the ACM Special Interest Group on Data Communication, ser. SIGCOMM '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 159–173.

[11] M. Mehic, M. Niemiec, S. Rass, J. Ma, M. Peev, A. Aguado, V. Martin, S. Schauer, A. Poppe, C. Pacher, and M. Voznak, "Quantum key distribution: A networking perspective," ACM Comput. Surv., vol. 53, no. 5, Sep. 2020.

[12] A. S. Cacciapuoti, M. Caleffi, F. Tafuri, F. S. Cataliotti, S. Gherardini, and G. Bianchi, "Quantum internet: Networking challenges in distributed quantum computing," IEEE Network, vol. 34, no. 1, pp. 137–143, 2020.

[13] L. Gyongyosi and S. Imre, "Entanglement availability differentiation service for the quantum internet," Scientific Reports, vol. 8, 2018.

[14] R. Van Meter, T. Satoh, T. D. Ladd, W. J. Munro, and K. Nemoto, "Path selection for quantum repeater networks," Networking Science, vol. 3, no. 1-4, p. 82–95, Dec 2013.

[15] S. Pirandola, R. Laurenza, C. Ottaviani, and L. Banchi, "End-to-end capacities of a quantum communication network," Communications Physics, vol. 2, 2019.

[16] ——, "Fundamental limits of repeaterless quantum communications," Nature Communications, vol. 8, 2017.

[17] S. Khatri, "Policies for elementary links in a quantum network," Quantum, vol. 5, p. 537, 2021.

[18] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski et al., "Human-level control through deep reinforcement learning," nature, vol. 518, no. 7540, pp. 529–533, 2015.

[19] J. S. Bell, "On the einstein podolsky rosen paradox," Physics Physique Fizika, vol. 1, no. 3, p. 195, 1964.

[20] J. Yin, Y. Cao, Y.-H. Li, S.-K. Liao, L. Zhang, J.-G. Ren, W.-Q. Cai, W.-Y. Liu, B. Li, H. Dai, G.-B. Li, Q.-M. Lu, Y.-H. Gong, Y. Xu, S.-L. Li, F.-Z. Li, Y.-Y. Yin, Z.-Q. Jiang, M. Li, J.-J. Jia, G. Ren, D. He, Y.-L. Zhou, X.-X. Zhang, N. Wang, X. Chang, Z.-C. Zhu, N.-L. Liu, Y.-A. Chen, C.-Y. Lu, R. Shu, C.-Z. Peng, J.-Y. Wang, and J.-W. Pan, "Satellite-based entanglement distribution over 1200 kilometers," Science, vol. 356, no. 6343, pp. 1140–1144, 2017. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.aan3211

[21] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," Journal of artificial intelligence research, vol. 4, pp. 237–285, 1996.

[22] C. J. Watkins and P. Dayan, "Q-learning," Machine learning, vol. 8, no. 3-4, pp. 279–292, 1992.

[23] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," nature, vol. 521, no. 7553, pp. 436–444, 2015.

[24] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," IEEE Signal Processing Magazine, vol. 34, no. 6, pp. 26–38, 2017.

[25] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in Proceedings of the 15th ACM workshop on hot topics in networks, 2016, pp. 50–56.
[26] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," IEEE Communications Surveys & Tutorials, vol. 21, no. 4, pp. 3133–3174, 2019.
[27] M. S. Frikha, S. M. Gammar, A. Lahmadi, and L. Andrey, "Reinforcement and deep reinforcement learning for wireless internet of things: A survey," Computer Communications, vol. 178, pp. 98–113, 2021.
[28] Y. L. Lee and D. Qin, "A survey on applications of deep reinforcement learning in resource management for 5g heterogeneous networks," in 2019 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC). IEEE, 2019, pp. 1856–1862.
[29] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," arXiv preprint arXiv:1508.04025, 2015.
[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in Advances in neural information processing systems, 2017, pp. 5998–6008.
[31] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," Machine learning, vol. 8, no. 3-4, pp. 293–321, 1992.
[32] L. Bottou, "Stochastic gradient descent tricks," in Neural networks: Tricks of the trade. Springer, 2012, pp. 421–436.
[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in Proceedings of the 3rd International Conference on Learning Representations, 2015.
[34] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," Nature, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2
[35] J. D. Hunter, "Matplotlib: A 2d graphics environment," Computing in Science & Engineering, vol. 9, no. 3, pp. 90–95, 2007.
[36] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep., 2008.
[37] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org.

LINH LE received the bachelor's degree in information technology from the Hanoi University of Science and Technology in 2013, the M.S. degree in information system from Marshall University in 2015, and the Ph.D. degree in analytics and data science from Kennesaw State University, GA, USA, in 2019. He is currently an Assistant Professor with the Department of Information Technology, College of Computing and Software Engineering, Kennesaw State University. He has publications in peer-reviewed journals, book chapters, and multiple conference proceedings, most notably, Neurocomputing journal, IEEE International Conference on Big Data, and a pending patent in a collaboration with Equifax Inc. He also has led several projects that were sponsored by industrial partners, including Kimberly Clark, Blue Ridge Global, Equifax, Alcon, and Cognira. His research focuses on deep learning and machine learning, particularly designing new techniques and applying them to solve critical problems, including cybersecurity, health informatic, manufacturing, and fintech.

TU N. NGUYEN (Senior Member, IEEE) received the Ph.D. degree in electronic engineering from the National Kaohsiung University of Science and Technology (formerly, National Kaohsiung University of Applied Sciences) in 2016. He is currently an Assistant Professor of computer science at Kennesaw State University, USA. Prior to joining KSU, he was an Assistant Professor of computer science at Purdue University Fort Wayne. He was a Post-Doctoral Associate with the Department of Computer Science and Engineering, University of Minnesota—Twin Cities, in 2017. Prior to joining the University of Minnesota, he worked as a Post-Doctoral Researcher with the Intelligent Systems Center, Missouri University of Science and Technology, in 2016. His research focuses on developing fundamental mathematical tools and principles to design and develop smart, secure, and self-organizing systems, with applications to network systems, cyber-physical systems, cybersecurity, and quantum communications. His research has resulted in more than 70 publications in leading academic journals as well as conferences, including IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, IEEE TRANSACTIONS ON COMPUTERS, IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, ACM Transactions on Internet Technology, IEEE TRANSACTIONS ON NETWORK SCIENCE AND ENGINEERING, IEEE TRANSACTIONS ON COGNITIVE COMMUNICATIONS AND NETWORKING, ACM Transactions on Sensor Networks, IEEE TRANSACTIONS ON POWER DELIVERY, and IEEE Conference on Local Computer Networks (LCN). He has served as a Technical Program Committee (TPC) Member for several international conferences, including IEEE INFOCOM, IEEE GLOBECOM, IEEE LCN, IEEE RFID, IEEE ICC, and IEEE WCNC. He is a NSF CRII Award Recipient of 2021. He has been in different organizing committees, such as being the TPC-chair and the general chair for several IEEE/ACM/Springer flagship conferences. He has engaged in many professional activities, including serving as an editor/guest editor for academic journals, such as an Associate Editor of IEEE SYSTEMS JOURNAL and IEEE ACCESS and a Leading Guest Editor of IEEE TRANSACTIONS ON COMPUTATIONAL SOCIAL SYSTEMS, IEEE Internet of Things Magazine, and IEEE JOURNAL OF BIOMEDICAL AND HEALTH INFORMATICS. He is also a Technical Editor of Computer Communications. He is the Editor-in-Chief of the book series: Advances in Distributed Computing and Blockchain Technologies.