ADVANCED QUANTUM TECHNOLOGIES
www.advquantumtech.com

Quantum Neural Networks for a Supply Chain Logistics Application

Randall Correll, Sean J. Weinberg, Fabio Sanches, Takanori Ide, and Takafumi Suzuki*

Problem instances of a size suitable for practical applications are not likely to be addressed during the noisy intermediate-scale quantum (NISQ) period with (almost) pure quantum algorithms. Hybrid classical-quantum algorithms have potential, however, to achieve good performance on much larger problem instances. One such hybrid algorithm on a problem of substantial importance: vehicle routing for supply chain logistics with multiple trucks and complex demand structure is investigated. Reinforcement learning with neural networks with embedded quantum circuits is used. In such neural networks, projecting high-dimensional feature vectors down to smaller vectors is necessary to accommodate restrictions on the number of qubits of NISQ hardware. However, a multi-head attention mechanism is used where, even in classical machine learning, such projections are natural and desirable. Data from the truck routing logistics of a company in the automotive sector is considered, and the methodology is applied by decomposing into small teams of trucks and results are found comparable to human truck assignment.

1. Introduction

In this work, we take explore a hybrid quantum-classical orthogonal neural network implemented in a reinforcement learning (RL) scheme to develop agents that can obtain good solutions in complex supply chain problems. In the setting of commercial operations, computational problems of substantial theoretical and practical difficulty regularly arise. Even a small improvement on the quality of solutions to such problems can translate to a very substantial benefit. One such problem, heavily studied in operations research, is the vehicle routing problem.[1,2]

In a previous effort,[3] we build on the work of ref. [4] by adding new techniques allowing for multiple trucks and for far more general requirements for trucks. While our model is specially designed with Aisin Corporation's vehicle routing problems in mind, our techniques for applying reinforcement learning to a very general class of routing problems apply widely.

Vehicle routing problems are NP-hard combinatorial optimization problems for which there are numerous heuristic algorithms which yield approximate solutions. Relatively recently, there has been interest in solving such routing problems using reinforcement learning.[5] In this context, a truck driving between nodes can be thought of as an agent performing actions (selecting its next node to drive to) in the environment of the supply chain.

The advent of quantum computing promises potential speedups in computation for a variety of problems. How well they apply to heuristic algorithms such as machine learning is an active area of research. Here we adapt quantum orthogonal neural networks as part of an attention head mechanism in a reinforcement learning scheme. The quantum circuit at the heart of the orthogonal neural network implemented in a supply chain logistics workflow is tested both using classical emulation and implementation on real quantum hardware. The experiments yields results which are preliminary but promising for the implications for quantum-boosted algorithms and applications as the underlying quantum algorithms and quantum computing hardware mature.

2. Quantum Orthogonal Neural Networks

There has been significant recent development in quantum algorithms for neural networks—so-called quantum neural networks (QNNs). This includes the theoretical[6] and experimental study[7,8] of how to best embed the data into the quantum circuit and maximize the separation of input data in Hilbert space. It also includes efforts to improve the training of QNNs, especially to avoid barren plateaus of parameterized circuits.[9,10] And aside from these abstract studies, QNNs have been applied to commercial and industrial applications to compare their performance against classical neural networks, assess their suitability

R. Correll, S. J. Weinberg, F. Sanches
QC Ware Corp.
195 Page Mill Road, Suite 113, Palo Alto, CA 94306, USA
E-mail: randall.correll@qcware.com
T. Ide
AISIN CORPORATION
Tokyo Research Center
1-18-13 Sotokanda, Chiyoda-ku, Tokyo 101-0021, Japan
T. Suzuki
Aisin Technical Center of America
2055 Gateway Place, Suite 410, San Jose, CA 95110, USA

The ORCID identification number(s) for the author(s) of this article can be found under https://doi.org/10.1002/qute.202200183

DOI: 10.1002/qute.202200183

for eventual use, and estimate their scalability in terms of problem size versus quantum computing hardware resources. Such efforts have explored QNNs for finance,[11] drug discovery and response,[12,13] automotive,[14] and medical imaging.[15] The last of these works explored the use of a quantum version of an orthogonal neural network.

Orthogonal neural networks (ONNs) are a type of neural network where linear layers are constrained to be orthogonal matrices.[16] Such neural networks have important advantages over neural networks with unconstrained weight matrices, especially in the case of deep neural networks; in this context, ONNs can avoid issues with generalization and small gradients.[16] However, classically computing the orthogonal layers is computationally expensive, so there is a trade-off of more work for better return.

Recently, quantum algorithms have been developed which perform the same computation as ONNs.[17] The unitary nature of quantum circuits naturally allow the computation of orthogonal layers with fewer computational steps than the classical computation. We briefly review the aspects of this work related to our models in this section.

The quantum ONN (QONN) algorithm of ref. [17] takes as input a normalized $\mathbf{x} \in \mathbf{R}^n$ with $\sum_{i=1}^{n} x_i^2 = 1$ and determines, to arbitrary precision, the result of applying a matrix $W \in \mathrm{SO}(n)$ acting on $\mathbf{x}$. Put differently, the algorithm rotates vectors on the $n-1$ sphere S^{n-1}. Because $\mathrm{SO}(n)$ is a Lie group of dimension $n(n-1)/2$, we require this number of parameters $\boldsymbol{\theta}$ to specify W, and these parameters are considered to be an input for the quantum algorithm.

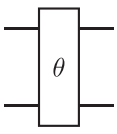
The algorithm for computing $W\mathbf{x}$ has three steps which we explain the subsections that follows:

1) Construct a quantum circuit that loads $\mathbf{x}$ with a unary encoding,
2) Append to the circuit a pyramidal quantum circuit with $n(n-1)/2$ gates parameterized by $\boldsymbol{\theta}$, and
3) Perform a tomography algorithm to recover the components of $W\mathbf{x}$.

Before explaining these steps further, we describe the two-qubit gate used throughout as a building block: the reconfigurable beam-splitter (RBS) gate. RBS gates are specified by one parameter θ. In the computational basis ordered as $|00\rangle, |01\rangle, |10\rangle, |11\rangle$, the RBS gate for a given θ is

$$\mathrm{RBS}(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{1}$$

In a quantum circuit diagram, we will use the notation



to denote RBS gates with parameter θ.

2.1. Unary Vector Encoding

Consider a vector $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbf{R}^n$. If $\mathbf{x}$ is normalized, we can consider the quantum state

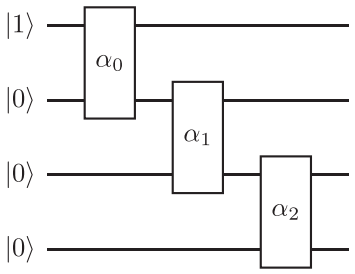
$$|\mathbf{x}\rangle = x_1|100\cdots00\rangle + x_2|010\cdots00\rangle + \cdots + x_n|000\cdots01\rangle. \tag{2}$$

This state encodes $\mathbf{x}$ in a simple way. The encoding is extremely sparse, involving only an n-dimensional subspace of the Hilbert space.

A key property of the RBS gate representing Equation (1) is that they map unary states to unary states. By a unary state, we mean any state which has the form of Equation (2). To see this, consider the action of an RBS gate on a unary basis vector $\psi_i = |0\ldots010\ldots0\rangle$. (We are using ψ_i to denote a computational basis vector which is 0 for all qubits except for the one in position i.) If the RBS gate acts on a pair of qubits a, b that does not include i, then it acts as the identity on ψ_i. On the other hand, if $a = i$ then the RBS gate results in a superposition of ψ_a and ψ_b, both of which are unary states.

In fact, we can make a stronger statement. An RBS gate, when acting on any unary state with the form of Equation (2) results in a new unary state $|x\rangle'$ which is obtained by applying an SO(2) operation on the 2-dimensional subspace corresponding to the qubits on which the RBS gate acts. From this it follows that any combination of RBS gates implements an orthogonal matrix in the unary encoding.

An explicit construction using only RBS gates for this encoding can be found in.[17] The procedure is to begin with the state $|100\ldots00\rangle$ and use a diagonal stack of $n-2$ RBS gates:

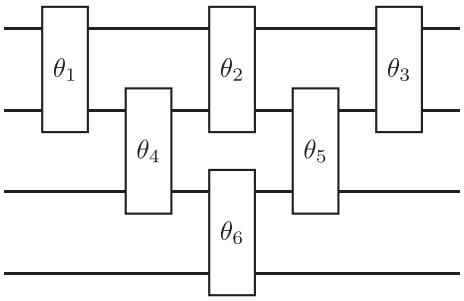


By an $O(n)$ classical computation, we can determine $\alpha_0, \ldots, \alpha_{n-2}$ which construct the desired loaded vector.

2.2. Pyramidal RBS Construction

In this section, we review the construction of[17] to explain how the state $|\mathbf{x}\rangle$ of Equation (2) can be fed into a simple quantum circuit, built entirely with RBS gates, to yield the state $|W\mathbf{x}\rangle$ where W is an orthogonal matrix and the notation, once again, implies a unary encoding of the resulting vector.

The circuit is a pyramid of RBS gates as follows in the $n = 4$ case:



As we explained in Section 2.1, RBS gates apply orthogonal matrices to unary states (in the unary encoding). Thus, we are guaranteed that this pyramidal circuit maps a unary state $|\mathbf{x}\rangle$ to another unary encoded state $|W\mathbf{x}\rangle$ where $W \in \mathrm{O}(n)$. Moreover, the pyramidal construction applies precisely $n(n-1)/2$ gates which matches the Lie group dimension of $\mathrm{O}(n)$. This strongly suggests that all matrices W in $\mathrm{SO}(n)$, the connected component of the identity of $\mathrm{O}(n)$, can be obtained by appropriately selecting $\boldsymbol{\theta}$. Construction of parameters to represent a desired matrix are given in.[17]

2.3. Tomography

With the state $|W\mathbf{x}\rangle$ constructed, we still need to determine the final output via tomography.[17] provides multiple approaches for this. In the experiments of this work, we adopted one of their techniques which we briefly review here for convenience.

Consider a unary-encoded state $|\mathbf{x}\rangle$. By measuring repeatedly, we obtain $p_1, \ldots, p_n$, the probabilities of measuring $\psi_1, \ldots, \psi_n$ respectively. These are estimates of $x_1^2, \ldots, x_n^2$ respectively.

The only information that we are missing is signs of $x_1, \ldots, x_n$. It is not possible to determine an overall sign, but if we assume $x_1 > 0$, the following procedure can be used to find all other signs. We append RBS gates with $\theta = \pi/4$ acting on qubits 1-2, qubits 3-4, 5-6, and so on until $(n-1) - n$ if n is even or $(n-2) - (n-1)$ otherwise. With this construction, measuring p_1 and p_2 correspond to $(x_1 + x_2)^2$ and $(x_1 - x_2)^2$. Thus, $p_1 > p_2$ if and only if x_1 and x_2 have different signs. We can then do the same construction with the RBS gates offset by one to determine if x_2 and x_3 have the same signs, and so on.

3. Vehicle Routing Problems

This section describes the mathematical problem to which we apply a hybrid classical-quantum machine learning model. The problem is a very general version of a vehicle routing problem. This routing problem was described in ref. [18] and, particularly, in ref. [3]. In this section, we give a succinct description of this general vehicle routing problem, and we refer the reader to ref. [3] for a detailed explanation.

The goal of the general vehicle routing problem is to give a problem statement which is very closely related to an actual supply chain logistics problem encountered by Aisin Corporation, a Japanese automotive component manufacturing company. The routing problem of Aisin Corp. is enormously complicated, involving many trucks and complex routing requirements for a large number of boxes. Our general vehicle routing problem includes the salient features of the Aisin Corp. logistics problem but makes small simplifications to make it possible to attack the problem with efficient machine learning techniques.

3.1. Basic Vehicle Routing Problem

Vehicle routing problems (VRPs)[1,2] come in many forms. One of the most basic problems starts with the following input:

- A collection of locations (nodes) and driving times between all nodes,
- one special node called a depot,
- a specified amount of *demand* to be picked up from each location and brought to the depot, and
- the capacity of a truck (i.e., the amount of demand that a truck can carry).

The routing problem then asks to find the minimal driving time path for a truck which starts at the depot and fulfills all demand requirements.

3.2. General Vehicle Routing Problem

There are two major features of the logistics problems of Aisin Corp that are not captured with basic vehicle routing problems: multiple trucks and tensor demand structure. The presence of multiple trucks is an obvious challenge that requires little explanation: solutions to multi-truck vehicle routing problems can involve subtle collaboration among trucks. Tensor demand structure is a concept described in refs.[3,18] which we now recount and expand upon.

In standard VRPs, there is a special depot node to which all demand must be brought. However, in a more realistic problem, requirements are more general. For example, we could have the requirement of bringing demand d from node 2 to node 3. Such a requirement can be stored as a component of a tensor: D_{23} We could also have a requirement like bringing demand first from node 5 to node 2 and then from node 2 to node 3. This can be stored as a component of a rank-3 tensor D_{523}.

Demand tensors can be off-board or on-board. The examples above are off-board demand because they describe material which is not on any truck and is waiting to be picked up. On-board demand refers to demand currently on a truck. For example, truck number m can have demand which needs to be dropped off at node i as its final stop. We can store this demand as contributing to a tensor E_i^m. If truck m has demand which needs to go to node i but then will need to go to node j (after being picked up again, perhaps by a different truck), then we can store this as a component to a rank 2 tensor as in E_{ij}^m.

Another situation that arises in realistic situations is *cyclic* off-board demand. This means, for example, that demand needs to go from node i to node j and then back to i. This can arise in several logistical setting. For Aisin Corp., this is a ubiquitous matter: boxes used to transport parts must be returned to their original departure location. Cyclic demand can also have higher rank: a box may need to go from node i to j, then j to k, and finally back to

Table 1. The table shows examples of demand structures with different indicial notation and explains how they should be interpreted in terms of where demand is picked up and/or dropped off.

Component	Description of most recent event
$D^{\text{cyclic}}_{3\,7\,4}$	Initial material at node 3 waiting to be picked up
$E^{m=2}_{7\,4\,3}$	Picked up from node 3 by truck 2, next stop node 7
$D_{7\,4\,3}$	Dropped off at node 7 by truck 2
$E^{m=1}_{4\,3}$	Picked up from node 7 by truck 1, next stop node 4
$D_{4\,3}$	Dropped off at node 4 by truck 1
$E^{m=3}_{3}$	Picked up from node 4 by truck 3, next stop node 3
0	Dropped off at node 3 by truck 3, requirements fulfilled

i. In situations where we have cyclic demand, we use the term *direct demand* to refer to demand which is not cyclic and we use notation like D^{cyclic}_{ijk} and D^{direct}_{ijk} to describe cyclic and direct demand tensors. Note that we could describe cyclic demand with a direct demand tensor of higher rank, but this is more computationally expensive in practice.

Various types of demands convert between each other as trucks pick up and drop off material. This can be understood through the following example shown in **Table 1** adapted from ref. [3] which gives events for cyclic initial demand starting at node 3 that must go to node 7 and then node 4 before returning to node 3.

An instance of the general vehicle routing problem is specified by providing an initial off-board demand tensors D, a collection of nodes $z_1, \ldots, z_n$, the number of trucks N, a matrix of driving times between nodes T_{ij}, and the capacities of trucks $C_1, \ldots, C_N$.

An important construction that we will use in this work is the concept of *myopic demand vectors*, given by

$$\epsilon^m_i = \sum_{i_2,\ldots,i_r} E^m_{i,i_2,\ldots,i_r} \tag{3}$$

and

$$\delta^{\text{out}}_i = \sum_{i_2,\ldots,i_r} D^{\text{total}}_{i,i_2,\ldots,i_r} \tag{4}$$

where D^{total} is the sum of cyclic and direct demand. In words, ϵ^m_i is the total material on truck m which needs to be dropped off at node i for its next stop, and δ^{out}_i is the total material at node i that is waiting to be picked up. We will also define an ingoing myopic off-board demand

$$\delta^{\text{in}}_i = \sum_{i_1,i_3,\ldots,i_r} D^{\text{total}}_{i_1,i,i_3,\ldots,i_r} \tag{5}$$

Of course, we could repeat this construction for any of the indices, but the further we go to the right in index, the less "myopic" the quantity is.

3.3. Relationship with Aisin Corp. Logistics

The general VRP that we described in Section 3.2 is inspired directly by the operations of Aisin Corp. in Japan. They are required, on a daily basis, to ship 340 000 boxes among 21 locations around Aichi Prefecture. These boxes have either rank-2 or rank-3 cyclic demand requirements: all boxes must be returned, and some have required intermediate stops (related to the need to hold at warehouses due to space limitations).

Despite the very large number of boxes, there is substantial redundancy. There are approximately 15 000 unique boxes, with the 340 000 including repetitions of identical boxes. Moreover, among the 15 000 boxes, many have identical routing requirements: there are only 107 unique routing requirements. We refer to the collection of all boxes with a specific routing requirement as a "box group." By summing over the volumes of all boxes within a box group, we can obtain a contribution to the initial cyclic rank-2 or rank-3 demand tensor. In this way, an instance of the general VRP can approximate an instance of the realistic logistics problem. Aisin typically requires about 150 trucks to complete the daily logistics assignments in the 16-h work day. These daily logistics assignments are produced by logistics experts with significant experience and using spreadsheet software to make a first estimate of routes, compute drive times, assess pick-up/drop-off assignments at each factory or depot node, and then iterate through that process until satisfactory solutions are obtained.

4. Hybrid Quantum-Classical Attention Mechanisms

In this section, we present a variation on multi-head attention mechanisms that incorporates the QONN algorithm described in Section 2. Our model is closely based on the encoder-decoder of ref. [4] as well the generalization of ref. [3]. In these works, an attention model is used to compute the policy of a reinforcement learning agent that learns to solve combinatorial optimization problem instances. We will use our model to solve the general vehicle routing problem discussed in Section 3.2.

4.1. Standard Attention Mechanism

The encoder and decoder of ref. [4] relies on a multi-head attention mechanism which we briefly review in this section. The encoder and decoder of ref. [3] modify this structure somewhat, and we review that modification here as well.

Starting with encoded nodes $\mathbf{h}_1, \ldots, \mathbf{h}_n \in \mathbf{R}^d$, we compute vectors known as queries, keys, and values. The queries and keys have a dimension α which is an arbitrary hyperparameter. In a single-head attention mechanism, we compute a single query, key, and value vector for each encoded node $\mathbf{h}_i$:

$$\mathbf{q}_i = M^{\text{query}}\,\mathbf{h}_i \in \mathbf{R}^\alpha \tag{6}$$

$$\mathbf{k}_i = M^{\text{key}}\,\mathbf{h}_i \in \mathbf{R}^\alpha \tag{7}$$

$$\mathbf{v}_i = M^{\text{value}}\,\mathbf{h}_i \in \mathbf{R}^d \tag{8}$$

Here, each M is a matrix mapping $\mathbf{R}^d$ to either $\mathbf{R}^d$ or $\mathbf{R}^\alpha$.

For the multi-head attention mechanism, we have a positive integer n_{heads}. For each $s \in \{1, \ldots n_{\text{heads}}\}$, we have a corresponding

query, key, and value map and thus different queries, keys, and values computed:

$$\mathbf{q}_{s\,i} = M_s^{\text{query}}\,\mathbf{h}_i \in \mathbf{R}^{\alpha}, \tag{9}$$

$$\mathbf{k}_{s\,i} = M_s^{\text{key}}\,\mathbf{h}_i \in \mathbf{R}^{\alpha}, \tag{10}$$

$$\mathbf{v}_{s\,i} = M_s^{\text{value}}\,\mathbf{h}_i \in \mathbf{R}^{\beta}. \tag{11}$$

All of the linear maps M are learned during training.

An important feature of this mechanism is that the node index i behaves as batch index: the same linear maps are applied to all of the nodes (although different maps are used for different attention heads s). This structure allows sequences to have arbitrary length rather than hard coding the sequence length.

An important reason that multi-head attention is used is to effectively reduce the dimensions of keys, queries, and values. When using a single attention head, we generally take $\alpha = d$ or at least $\alpha \approx d$. However, with multi-head attention we can used $\alpha, \beta \approx d/n_{\text{heads}}$. As we will discuss in Section 4.2, having a smaller α, β makes it possible to integrate the QONNs of Section 2 in a way that can feasibly be executed on noisy intermediate-scale quantum (NISQ) hardware.

The next step is to compute a *compatibility* for every query-key pair (for each head). We define

$$u_{s\,ia} = \frac{1}{\sqrt{\alpha}}\mathbf{q}_{s\,i} \cdot \mathbf{k}_{s\,a} \tag{12}$$

with $\cdot$ denoting a standard dot product. After this, we compute

$$\rho_{s\,ia} = \frac{\exp(u_{s\,ia})}{\sum_b \exp(u_{s\,ib})} \in \mathbf{R} \tag{13}$$

which is used to weight a sum over values

$$\mathbf{h}'_{s\,i} = \sum_a \rho_{s\,ia} v_{s\,a}. \tag{14}$$

The only remaining step to to merge the data from the n_{heads} attention heads. To do this, we simply concatenate the output from each head,

$$\mathbf{h}'_{1\,i} \oplus \ldots \oplus \mathbf{h}'_{n_{\text{heads}}\,i} \tag{15}$$

and use a learned linear map with bias on this vector, from dimension $n_{\text{heads}}\beta$ to dimension d to recover a vector $\mathbf{h}'_i \in \mathbf{R}^d$. Here, the symbol $\oplus$ denotes direct sum, which is equivalent to concatenation in this context.

We use the symbol MHA to denote the function which starts with encoded vectors $\mathbf{h}_i \in \mathbf{R}^d$, and returns the output sequence $\mathbf{h}'_i \in \mathbf{R}^d$. MHA computes queries, keys, and values following Equations (9)-(11), then computes compatibility and new encoded vectors for each head, and finally maps to the original encoding dimension.

An important technique used in both refs. [[4]] and[[3]] is the concept of source terms. Source terms allow us to modify key, query, and values in a way that depends on some other quantity, a "source". Roughly speaking, the idea is to modify, for example, keys from $\mathbf{k}_i = M\mathbf{h}_i$ to

$$\mathbf{k}_i = M\mathbf{h}_i + \phi_i\mathbf{u} \tag{16}$$

where ϕ_i is a real number for each i and $\mathbf{u}$ is a learned vector with dimension α, the dimension of keys. The quantity ϕ_i is referred to as a vector source because it has one index i. In general, we can also have an arbitrary number of sources by summing over different learned vectors.

As a more concrete of sources, consider the myopic demands $\delta_i^{\text{out}}, \delta_i^{\text{in}}$ of Equations (4) and (5). We can use them to modify Equations (10) and (11)

$$\mathbf{k}_{s\,i} = M_s^{\text{key}}\,\mathbf{h}_i + \mathbf{u}_s^{\text{key,out}}\,\delta_i^{\text{out}} + \mathbf{u}_s^{\text{key,in}}\,\delta_i^{\text{in}}, \tag{17}$$

$$\mathbf{v}_{s\,i} = M_s^{\text{value}}\,\mathbf{h}_i + \mathbf{u}_s^{\text{val,out}}\,\delta_i^{\text{out}} + \mathbf{u}_s^{\text{val,in}}\,\delta_i^{\text{in}}. \tag{18}$$

Here, s is an index for attention heads, and various $\mathbf{u}_s$ are learned vectors with the same dimension as the left hand side of the equations they appear in; for example, $\mathbf{u}_s^{\text{key,out}}$ is a vector with the same dimension as keys which is α as specified in Equation (10). Note that in this example, two sources are used for both keys and values.

4.2. Attention With Quantum Layers

There are a variety of ways that quantum neural circuits can play a role in attention mechanisms. One approach, developed in,ref. [19] is to associate a quantum state for each key and query, and to then apply a unitary operation for each key-query pair and to perform a measurement to obtain a quantum version of key-query compatibility, replacing Equation (12).

In this work, we consider instead a QONN construction. Consider keys, queries, and values obtained through Equation (10)-(11). For definiteness, we focus on keys $\mathbf{k}_{s\,i}$ which we take to have dimension α for each head s and each node index i.

Now we can apply the QONN algorithm of Section 2 to $\mathbf{k}_{s\,i}$. The particular ONN construction relies on parameters θ which specify the quantum circuit. We will use different parameters for different attention heads s, but, as usual, the parameters do not depend on i. Let L_s denote the operator which takes as input $\mathbf{k}_{s\,i}$ (for fixed s and i), applies the algorithm described in Section 2, including loading and tomography, and outputs the resulting classical vector found through tomography. Note that L_s constructs an α-qubit quantum circuit.

The usage of quantum circuits in individual attention heads is very well-suited for NISQ quantum hardware. As discussed above, when we use n_{heads} attention heads, it is most natural to use

$$\alpha, \beta \approx \frac{d}{n_{\text{heads}}} \tag{19}$$

The circuit construction uses α (or β for values) qubits, and the reduction in the number of qubits allows for implementation on hardware with limited size while not sacrificing the encoding dimension d. For example, with $d = 128$ and 16 attention heads, we require only 8 qubit circuits.

Additionally, in a model with n_{heads} attention heads, we have a the versatility of being able to select only a subset of heads in which to include QONNs. Thus, we can have non-negative integers $n_{\text{classical}}$ and n_{quantum} with $n_{\text{classical}} + n_{\text{quantum}} = n_{\text{heads}}$. In general, we have the freedom of choosing these integers differently for every key, query, and value in all attention heads throughout the encoder or decoder.

Tuning the number of quantum heads in various attention layers is of critical importance when attempting experiments involving near-term quantum hardware. The reason for this is not only limitations due to noisy hardware. Another major reason relates to access availability and monetary cost. When submitting jobs to quantum hardware, time delays can occur due to jobs queuing. The more separate quantum circuits that need to be prepared in a workflow, the more serious this consideration is. To execute an instance of the general VRP with a trained encoder–decoder model including quantum attention heads, the number of quantum circuits that must be prepared and measured can be very large.

4.3. Incorporating Tensor Structure

We now turn to an element of our attention mechanism which is unrelated to quantum algorithms. A major challenge for attention mechanisms is the incorporation of tensor structure like the tensor demand structure discussed in Section 3.2. The issue can be seen through Equation (16): we can convey to an attention mechanism vector quantities which are associated with a single node (ϕ_i is associated with node i in Equation (16)), but we cannot convey tensor quantities that are associated with multiple nodes.

In ref. [3], two approaches for incorporating tensor demand structure were provided. The most powerful method is a *tensor attention mechanism* where a single query $\mathbf{q}_i$ is used to probe a sequence of nodes, like (j, k, l). From the sequence (j, k, l), we obtain a key tensor K_{jkl}, which is, for each (j, k, l), a vector with dimension α, so we can compute a compatibility between node i and nodes (j, k, l). By also constructing value tensors, we essentially follow Equations (10) through (14). The power of the tensor attention mechanism is that it allows for tensor sources. However, to construct keys and value for sequences of tensors, we are forced to map not from $\mathbf{h}_i$ but instead from $\mathbf{h}_i \oplus \mathbf{h}_j \oplus \mathbf{h}_k$, incurring a substantial memory cost in practice.

While we maintain that a tensor attention mechanism is the natural approach when dealing with combinatorial optimization problems with tensor structure, we can often get away with only including myopic data, such as Equations (4)-(5). One step further is to consider matrix myopic off-board demand. In the case where we have rank 2 and rank 3 cyclic and direct off-board demand, this means

$$D_{ij} = D^{\text{total}}_{ij} + \sum_k D^{\text{total}}_{ijk} \tag{20}$$

where D^{total} is the sum of cyclic and direct off-board demand.

We can incorporate this matrix into the attention mechanism by using it to modify compatibility. We replace the dot product with

$$\frac{1}{\sqrt{\alpha}} G_{ij}\, \mathbf{q}_i \cdot \mathbf{k}_j \tag{21}$$

where G is some tensor determined by D. This approach can exaggerate query-key compatibility in cases where D_{ij} is large and suppress compatibility when the demand is small.

There are a few reasonable choices for G. The first is $G_{ij} = 1$, which reduces to a basic dot product compatibility. Next is $G_{ij} = M_{ij}$ where M is the mask defined as $M_{ij} = 1$ when $D_{ij} > 0$ and $M_{ij} = -\infty$ otherwise. Both of these are within the methodology of ref. [4]. A third and more novel choice of G is $G_{ij} = \log D_{ij}$. This last form has several virtues: it reduces to a mask in the sense that it approaches $-\infty$ as $Dij \to 0+$. Moreover, it can exaggerate compatibility when D_{ij} is large. A simple additional adjustment is to use

$$G_{ij} = AD_{ij} + B \log D_{ij}, \tag{22}$$

which is more sensitive to changes in D_{ij} for larger values.

Rather than having to pick from these various choices, we can choose all of them by taking advantage of the multiple heads. In other words, for a given head $s \in \{1, \ldots, n_{\text{heads}}\}$, we can put

$$G^s_{ij} = A^s_{\text{basic}} + A^s_{\text{mask}} M_{ij} + A^s_{\text{log}} \log D_{ij} + A^s_{\text{lin}} D_{ij} \tag{23}$$

5. Reinforcement Learning and the General VRP

We now describe how the ideas of the previous sections can be put together to construct a reinforcement learning policy.

Reinforcement learning concerns Markov decision processes where an agent performs an action when presented with the state of its environment and, as a stochastic result of the action, the state changes and the agent receives a reward. The action performed by the agent is determined by the agent's policy which is a function π that, given the state s, gives the probability $\pi(a \mid s)$ of performing a given action a .

In the context of combinatorial routing problems like the traveling salesman problem, the action of an agent would be the choice of which node to travel to next and the environment state would be the prior route. For a vehicle routing problem with a single truck, the situation is quite similar except that we also need to include in the state all prior information about the current demand structure:

$$\pi\left(z \mid (\xi_0, \ldots, \xi_{t-1}), D_{t-1}\right) \tag{24}$$

where ξ denotes the prior route and z is a proposed node to drive to next. Note that we are using D_{t-1} loosely to include all relevant information about demand.

For problems like this, the probability of choosing an entire route ξ is a product of probabilities at each step as long as the policy is fixed during the episode

$$\pi(\xi \mid D_0) = \prod_{t=1}^{k} \pi\left(\xi_t \mid (\xi_0, \ldots, \xi_{t-1}), D_{t-1}\right) \tag{25}$$

This structure is important because we will use the REINFORCE algorithm[20] which depends on the logarithm of the probability of choosing an action. The multiplication in Equation (25) converts to a summation, allowing us to deal with the probability of the entire route without it being too small to work with in practice.

In addition, the general VRP expands beyond a policy for a single truck and must account for other trucks as part of the environment, thus enabling the policy benefit from cooperation between multiple trucks. We have implemented such a general VRP model by devising a scheme that applies to teams of trucks in which at any given moment in time one truck is the *active* truck and the other *passive* trucks are part of the environment. The training proceeds through epochs of time in which the effects of the other trucks come into play. The overarching approach is described below.

5.1. Encoder–Decoder Policy

The machine learning model that we use as a model for π is an encoder-decoder attention model closely based on refs. [[3,4]]. In fact, we use almost the exact model of ref. [3] except for the role of quantum circuits. We therefore refer the reader to ref. [3] for the details of the encoder and decoder layers and only highlight the hyperparameters and the use of QONNs here.

The basic idea of the policy model is that the nodes $z_1, \dots, z_n$ along with information about the initial demand can be thought of as a sequence of data. We start with coordinates of the nodes, $\mathbf{x}_i \in \mathbf{R}^2$, which we take to lie in the unit square. We append information about initial demand structure

$$\bar{\mathbf{x}}_i = \mathbf{x}_i \oplus \left(\delta_i^{\text{in,init}}, \delta_i^{\text{out,init}}\right) \tag{26}$$

and use the sequence $(\bar{\mathbf{x}}_1, \dots \bar{\mathbf{x}}_n)$ as the input to an encoder. The encoder treats i, the node index, as a batch dimension.

Encoding proceeds through layers, and each encoder layer contains an attention sub-layer (Equation (27)) followed by a feed-forward sub-layer (Equation (28))

$$\tilde{\mathbf{h}}_i^{l-1} = \text{BN}\left(\mathbf{h}_i^{l-1} + \text{MHA}\left(\mathbf{h}_i^{l-1}\right)\right) \tag{27}$$

$$\mathbf{h}_i^{l} = \text{BN}\left(\tilde{\mathbf{h}}_i^{l-1} + \text{FF}\left(\tilde{\mathbf{h}}_i^{l-1}\right)\right) \tag{28}$$

In these equations, BN is a batch normalization layer,[21] FF is a feed-forward network, and MHA is a multi-head attention layer as described in Section 4.1 along with dynamical masking from 4.3. The feed-forward layers have a single hidden dimension d_{ff} and consist of a linear layer with bias mapping $\mathbf{R}^d \to \mathbf{R}^{d_{\text{ff}}}$ followed by a ReLU activation function, a dropout layer, and finally a linear map with bias back to $\mathbf{R}^d$.

The output of the final encoder layer gives encoded nodes $\mathbf{h}_i \in \mathbf{R}^d$. These nodes, along with contextual information about the current environment state, are then decoded. Decoding yields a probability distribution over possible nodes for the active truck to select as its next destination.

Our encoder and decoder structure are identical to that of ref. [3] other than the usage of quantum circuits. The encoder consists of three attention layers. As in ref. [3] we use dynamical masking (Equation (23)) rather than the tensor attention mechanism due to memory limitations.

Regarding the inclusion of quantum circuits, we use two different models:

1) *Simulation-only model*: A quantum orthogonal neural network is used for every key, query, and value in the encoder and decoder including all three encoding layers. We use $d = 128$ for encoding and 8 attention heads. There are 16 qubits for circuits with $\alpha = d/8$.
2) *Hardware experiment model*: We only use a quantum orthogonal neural network for queries and keys in the encoder. We do not use quantum circuits for the decoder. We use $d = 64$ and eight attention heads with 8 qubits per head.

Algorithm 1 REINFORCE variant for VRP.

Input: Parameterized policy π
Input: Integers `num_epochs, batch_size, batches_per_epoch`
Input: Initial parameter θ
$\theta_{\text{BL}} \leftarrow \theta$
for $e = 1, \dots,$ `num_epochs` **do**
  for $b = 1, \dots,$ `batches_per_epoch` **do**
    $\xi \leftarrow$ (`batch_size` many episodes from $\pi(\theta)$)
    $\xi_{\text{BL}} \leftarrow$ (`batch_size` many episodes from $\pi(\theta_{\text{BL}})$)
    $\nabla J \leftarrow \texttt{batch\_mean}((L(\xi) - L(\xi_{\text{BL}}))\nabla_\theta \log(\sum_{i=1}^{k} \pi(\xi^i, \theta)))$
    $\theta \leftarrow \text{descent}(\theta, \nabla J(\theta))$
  end for
  if `baseline_test()` **then**
    $\theta_{\text{BL}} \leftarrow \theta$
  end if
end for

5.2. Training and Simulation

For the purposes of training, our model was written entirely in PyTorch. In particular, quantum circuits are also emulated through PyTorch. The quantum circuit emulation approach is direct state vector manipulation: we load a unary state by hand, stored as a PyTorch `Tensor`, and we apply gates through matrix multiplication on appropriate subsystems. The final readout is also done in an nonphysical fashion: we simply read the unary components final state vector.

This approach has a major advantage of being easy to run efficiently for fewer than ≈ 30 qubits (beyond which point, the memory cost of storing state vectors becomes problematic). One disadvantage, however, is that we do not account for stochastic behavior of quantum computing, either due to error or simply because of the probabilistic aspect of measurement.

5.3. Learning Algorithm

Following refs. [[4] and[22]] we implemented a variant of REINFORCE[20] but now adapted for quantum circuits. Our REINFORCE variant is given in algorithm 1 below and is more fully described in ref. [3]. Note that this algorithm is broken up into epochs and batches.

For our purposes we take a variant of REINFORCE adapted to our vehicle routing problem as follows. First, for the purposes of the algorithm, we take the entire episode to be defined by a single action. In other words, the episode is just a_0, r_1. The action a_0 is the entire route specification $a_0 = (\xi_1, \xi_2, \dots \xi_k)$ where each

ξ_i are nodes. The reward r_1 is simply the negation of the route length (or time) $-L(\xi)$ which is computed by summing the appropriate distances between nodes based on a metric or on known travel times.

The second important aspect of our variant is the baseline methodology. This idea is roughly adapted directly from ref. [4]. This essentially means that some function b of states (but not of actions) is constructed with each episode and the return G in the algorithm is replaced by $G - b(s)$. This algorithm still converges to the optimal policy theoretically and, with a well-chosen baseline, does so much faster. In this work, we maintain a "baseline agent" which uses the same parameterized policy but does constantly update its parameter θ. Instead, the baseline agent uses an outdated parameter θ_{BL} which is occasionally updated to match the primary agent's θ, but only when the agent substantially and consistently outperforms the baseline.

The summation $\sum_{i=1}^{k} \pi(\xi^i, \theta)$ is a sum over the probabilities computed by the encoder/decoder network at each stage of the route. k refers to the number of steps in the route and the index i runs over steps in the route, not over batch entries. The entire computation is performed for each batch entry and averaged over.

6. Supply Chain Management Workflow

While the experimental approach of Section 5.1 is restricted to a modest system size, we can use the simulation-only model of Section 5.1 to test our model at a more ambitious scale. In fact, through a decomposition technique of breaking the full supply chain problem into subsets of a few trucks at a time, we can deploy our model on a full-scale problem to meet the demands of a commercial supply chain.

We thus turn to the problem of finding solutions to the Aisin Corp. vehicle routing problem instance (which we refer to as the AVRP).

6.1. Node Subset Search

Consider an instance of the general VRP with n nodes and demand structure D^{init}. Given the demand structure, the time matrix, and the driving window $T_{\max}$, we can estimate the number of trucks N that will be necessary to fulfill all requirements.

Suppose that we have an algorithm to find solutions to general VRPs with a smaller number of nodes and trucks and with smaller demand structure. Our algorithm works for $n' < n$ nodes and $N' < N$ trucks.

This situation arises in our context naturally: we can train, for instance, an agent to solve general VRP instances with ten nodes and three trucks, a scale smaller than the AVRP with its 21 nodes and over 100 trucks.

We should be able to use our smaller-scale algorithm to solve the larger problem by applying it repeatedly to different subsets to nodes to gradually fulfill all demand requirements. This raises a question of how to find good subsets.

We begin by looking at the demand structure D^{init}. For simplicity, assume that this consists only of rank-3 cyclic demand (other cases are very similar and we describe the necessarily modifications below). From D^{init} we can identify the nonzero components. These are tuples of nodes

$$\begin{aligned} &i_1 j_1, k_1 \\ &i_2 j_2, k_2 \\ &\quad\vdots \\ &i_u j_u, k_u \end{aligned} \tag{29}$$

where u is some integer (which happens to be 107 for the AVRP, corresponding to the 107 unique routing requirements). We can begin our node search by uniformly randomly selecting one of these triples of nodes from the list. (In cases where demand also includes rank 2 or another rank, we include tuples of appropriate length in the list for nonzero demand cases and we allow such tuples to be selected as well.) After drawing a tuple from the list, we remove it from the list. Suppose that we select the tuple (i_3, j_3, k_3). We define a starting subset of nodes as $A = \{i_3, j_3, k_3\}$. If $3 < n'$, we continue to draw nodes. Suppose that we next draw (i_5, j_5, k_5). We now consider the set $A = \{i_3, j_3, k_3, i_5, j_5, k_5\}$. If any node repeats (for instance, if $i_3 = j_5$), that entry is only counted once in the set. There will now be between 4 and 6 elements in A. If $|A| < n'$, we continue and otherwise we stop. Continuing in this way, we can either eventually run out of tuples or we can reach $|A| \geq n'$. If we reach $|A| = n'$, we stop and use A as our first guess of a node subset. If $|A|$ exceeds n', we remove the most recently added subset. If we run out of tuples, we stop.

After this process, we might have $|A| < n'$. In this case, we simply randomly add additional nodes outside of $|A|$ until reaching $|A| = n'$.

At this point, we have obtained a random node subset A. We repeat this process K times to obtain $k_{\text{node draws}}$ different random subsets, and we apply our algorithm $k_{\text{subset attempts}}$ times for each of the $k_{\text{node draws}}$ subsets. We compute the mean demand fulfilled for the $k_{\text{subset attempts}}$ trials, and we select the node subset with the highest mean demand coverage.

6.2. Execution Loop

With a technique for selecting node subsets, we are now in a position to describe the execution procedure. This begins with the initial demand structure D^{init} which is determined by the AVRP's requirements. Next, a node subset is selected through the node search technique of Section 6.1. We identify the portion of D^{init} that is supported by the subset and we map it onto a demand structure D' for n' nodes. To regulate the policy input, we clip D' at some maximum value C. We then apply the trained agent $k_{\text{execution trials}}$ times and select the trial with the highest percentage of covered demand. This trial has a specific routing for N' trucks and corresponds to a certain demand fulfillment. The route as well as the on-board and off-board demand at each step is saved and the initial demand D^{init} is modified: it is reduced by the amount of demand satisfied by the route.

This process is repeated, each time first performing node selection and then finding the best route out of $k_{\text{execution trials}}$ attempts. We repeat until all demand is satisfied. The total number of iterations of this procedure multiplied by N' will be the total number of trucks needed for our solution.

6.3. Simulated Execution

The result of the execution loop yields approximate solutions to the general VRP. We obtain a collection of routes for various trucks $x_{m\tau}$ where m ranges over trucks and τ over time steps for each truck. However, we still need a way to convert such routes to candidate solutions to the overall AVRP.

Our strategy is to interpret the routes x as "suggested routes" and to attempt to use them in a *full-scale* supply chain simulation. We make use of the simulation of ref. [3]. Every box is individually tracked, has rank-2 or rank-3 requirements, and we require that demand is cyclic—boxes must be returned to their origin node. Each box has a specific individual volume. The simulation is designed to be as similar as possible to the actual commercial routing problem of Aisin Corporation.

Trucks follow along the routes $x_{m,\tau}$ that they are assigned from the execution algorithm and they pick up boxes as is appropriate for their route. Boxes are picked up according to the algorithm described in ref. [18] and we carefully ensure that trucks only drive within the allowed drive time windows (two 8-h shifts). The result of full-scale simulation is that, rather than having a list of suggested truck routes, we obtain a precise statement about what each truck in the supply chain is doing at all times, including exactly which boxes must be picked up and dropped off at various nodes.

7. Results: Experiment With Ion Trap Quantum Hardware

In ref. [3] we implemented the entire supply chain management workflow classically using standard machine learning algorithms. In this work, we implement and test the workflow using quantum circuits of the ONNs implemented on real quantum computing hardware. First, we return our focus to the discussion of the quantum circuits in Section 2 and discuss the essential experiments needed to test their performance. Then we discuss the implementation of the neural network and attentions heads. And finally we discuss the performance of the quantum circuit in generating solutions when used as part of the overall workflow.

7.1. Hardware Experiment for QONN Accuracy

Current quantum hardware is limited by noise and the precision of the gates they can implement. We have a variety of quantum hardware to choose from and all have different strengths. There are different qubit technologies, and of these, superconducting and trapped ion are currently the most readily available and mature in the sense of implementing a universal gate set that would be appropriate for our quantum circuit. The number of available qubits is not the most relevant attribute as most experiments conducted to date by various researchers have been limited to 20 qubits or less due to limitations in noise and error in the quantum processor. Superconducting quantum processors have the advantage of much faster gate speeds (on the order of GHz) compared to trapped ion quantum processors (on the order of KHz to MHz). However, trapped ion processors have all-to-all connectivity which facilitates shorter depth circuits than the superconducting processors, which are limited to nearest neighbor connectivity.

The IonQ device was chosen because it has relatively good noise levels and the inherent full-connectivity of the qubits simplifies the embedding of the algorithmic circuit onto the hardware graph, thereby avoiding the need for additional gates operations to implement gate interactions between qubits that are not directly connected physically. Additionally, the IonQ device is commercially available on Amazon Web Services Braket cloud-based quantum platform, and this makes it available to access by the authors and other researchers via a straightforward subscription service.[23] Its main downside at this point is relatively slow gate speeds. That affects the prefactor in the scaling of the circuit computation and is readily accounted for when considering future hardware requirements and in comparison to superconducting qubits. Despite the emerging commercial availability to quantum computing hardware, it is still an expensive resource, so users must be careful to tune their algorithms to get the best performance for an affordable cost.

Having chosen IonQ to run the experiment, there is still an important question: how accurate is the QONN algorithm when run on IonQ? To answer this, we performed the following experiment using the 11-qubit IonQ device (this experiment could easily be replicated on other quantum hardware to test suitability for use in the full experiment).

For various number of qubits (4, 5, 6, 7, 8, 9, and 10 qubits), we constructed the pyramidal RBS circuit of Section 2.2. We used QC Ware's commercially available implementation of unary data loader circuits, and we used the tomography method described in Section 2.3. We used randomly selected RBS parameters and also randomly selected input vectors. All parameters and vectors were randomly drawn from a normal distribution with mean 1 and variance 1. For each fixed number of qubits (4, 5, 6, 7, 8, 9, and 10), we selected ten different randomly drawn parameter sets. We performed 500 measurements on each of the three circuits required for unary tomography. We therefore ran 210 quantum circuits with 500 measurements for each, a total of 105 000 measurements.

A sample of our results are shown in **Figures 1–4** for the cases of 4, 6, 8, and 10 qubits. In figure displays a montage of four runs selected to show the variety of results returned from each tomography measurement accomplished according to the parameters described in the preceding paragraph. Since the circuits are random, each of the four tomography results will look different. However the error is revealed in comparing the hardware (blue) and classically emulated (orange) pairs for each qubit. For an N-qubit tomography experiment, you will see N pairs of bars. In each pair we compare the hardware results against the classically computed emulation of the circuit. Where the hardware results (blue) differ from the emulation results (orange), the hardware results are in error.

In Figure 1, the 4-qubit case, there is generally good agreement between all pairs with an occasional sign discrepancy. As we include more qubits in the ONN, one can see evermore degraded performance as the noise and imprecision accumulate to corrupt the result, so that by the 10-qubit case, the hardware results have become almost completely random, averaging to zero.

Our insights are as follows:

- For up to 6 qubits, the performance of IonQ is quite good.

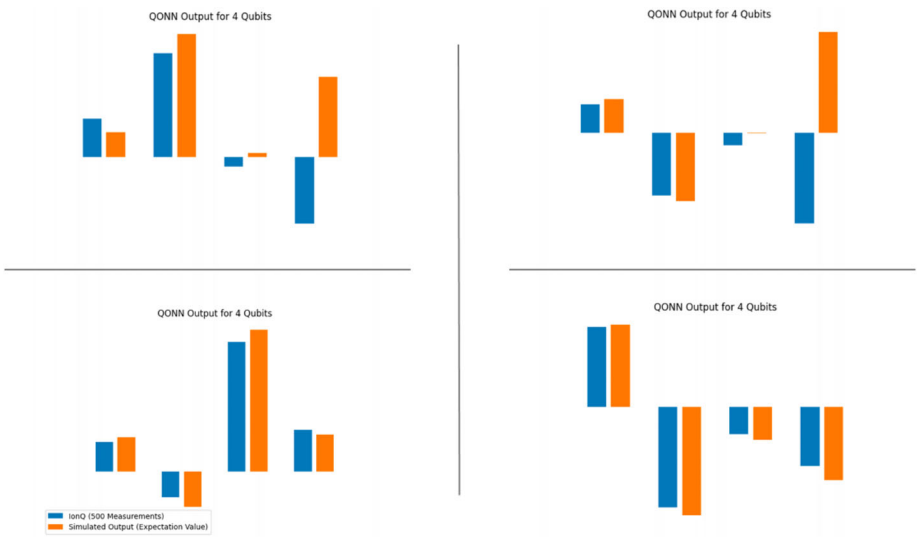


Figure 1. Results of 4-qubit QONN tomography.

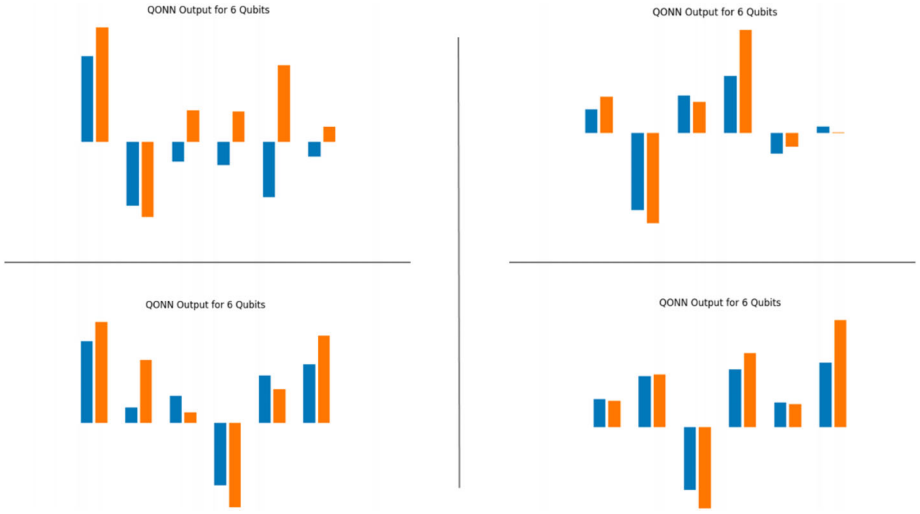


Figure 2. Results of 6-qubit QONN tomography.

- For up to 8 qubits, the performance of IonQ is sufficient to capture aspects of the desired QONN behavior.
- At and beyond 9 qubits, we found that QONNs are not appropriately captured by IonQ with its current capabilities.
- Occasionally signs out outputs can be flipped. This is a tomography error. Such errors can cause a cascade of errors: if the wrong sign for qubit 3 is selected, then we get the wrong signs for qubits 4, 5, 6, ... unless another error corrects the original error.
- We emphasize that the errors in quantum computing we are seeing here are not surprising and reflect the early state of quantum hardware.

7.2. Results: AVRP Training with Ion Trap Quantum Hardware

After testing and diagnosing the maximum number of qubits that could be reliably used in the QONN, we then tested how well the QONN can support a hybrid the entire AVRP workflow. First we needed to select a problem size suitable for the quantum hardware. We selected eight of the most active nodes from actual Aisin Group data. Specifically, we selected the following nodes:

1. Nishio cross-docking
2. No.1 and 2 plant
3. Okazaki and electric plant
4. Okazaki east plant
5. Tahara plant
6. GamagoriI plant
7. Kira plant
8. Meiko

These nodes actually have cyclic rank-2 and 3 demand that flows between them. Due to limitations of current quantum hardware, we need to keep the problem fairly small for the eventual

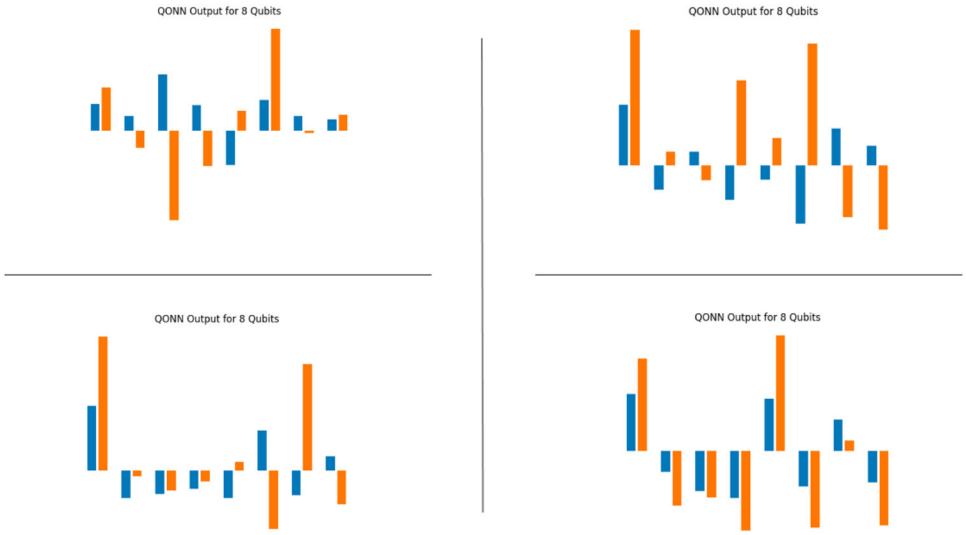


Figure 3. Results of 8-qubit QONN tomography.

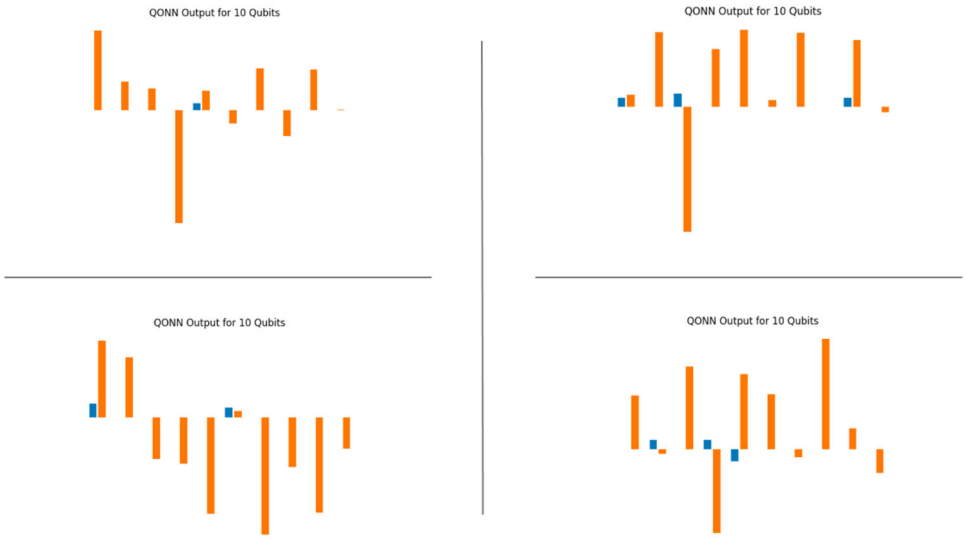


Figure 4. Results of 10-qubit QONN tomography.

hardware runs, so we choose to only consider direct rank-2 demand even for the classical neural network or the emulated quantum circuits. We sum over all demand and use the summation as a contribution to an effective rank-2 demand. We then drop the box return constraint. When hardware becomes more advanced, we should be able to include more constraints.

We know from the results of Section 7.1 that at the 8 qubit scale, we are pushing the limits of current hardware for QONN implementation. For this 8-node problem scenario, we trained a variety of models, classically or with our classical quantum circuit emulator, with slightly different problem parameters to study and compare the performance.

In the following examples in this section, keep in mind that *classically* refers to a completely classical neural network or layer, and *quantum* refers to a quantum circuit or neural network layer that we compute using our classical emulator. So, all examples in this section are computed on classical hardware, whether they are computing a traditional classical neural network or computing a quantum circuit via an emulator. And since the emulator does not mimic noise or randomness of the quantum circuit, the results of the *classical* and *quantum* computations for the same input data are the same.[17]

Now we show a series of examples with different parameters for number of trucks, nodes, demand structure, and box return constraints to illustrate the performance of the neural network and identify where the computational challenges lie.

First we show the result from a classical training run to demonstrate training on relatively large problem (more trucks and the most complicated constraints). **Figure 5** shows the classical training results versus training epoch for an 8-node, 3-truck, including rank 2 and rank 3 demand structure, and with cyclic box-return constraint. The cost is a dimensionless cost function, the cover-

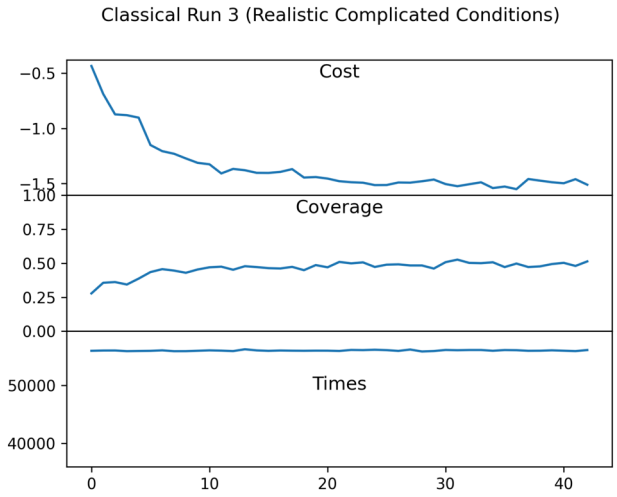


Figure 5. Classical training run. Highly realistic 3 truck, rank 2 and 3 demand, with cyclic box-return constraint. Uses 8 nodes.

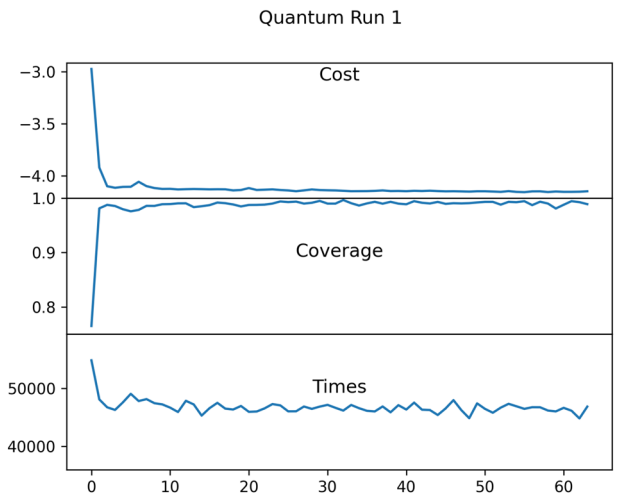


Figure 6. Quantum training run. 2 truck, rank 2 demand, with no cyclic box-return constraint. Uses 8 nodes.

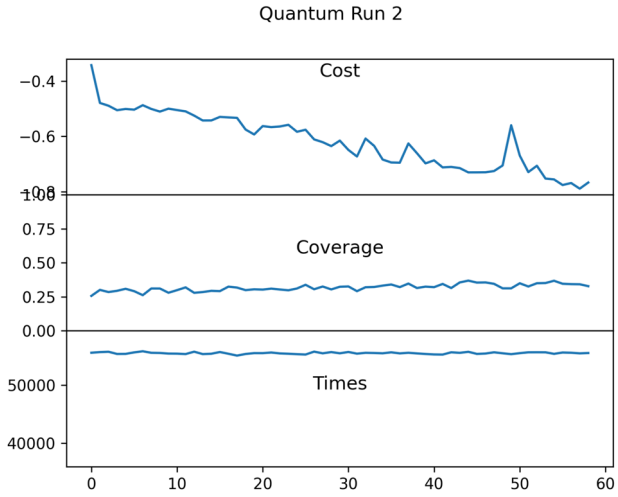


Figure 7. Quantum training run. 2 truck, rank 2 and rank 3 demand, with no cyclic box-return constraint. Uses 8 nodes.

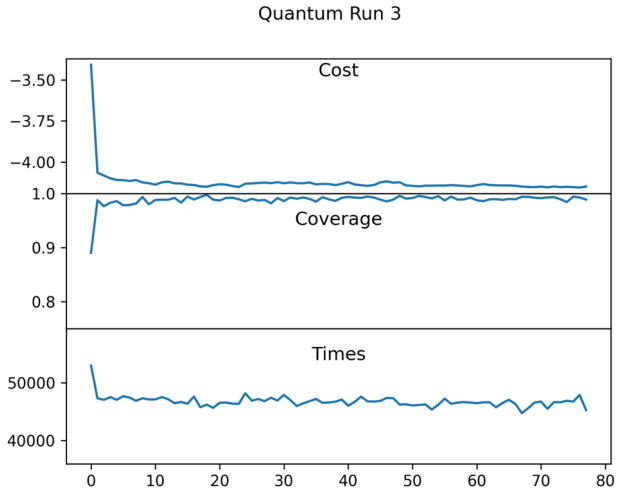


Figure 8. Quantum training run. 2 truck, rank 2 and rank 3 demand, with no cyclic box-return constraint. Uses 8 nodes.

age is a normalized variable of the total demand (1.00 = 100%), and the Times represent how much time in seconds was used to complete the problem (must be less than the 16 h = 57 600 s constraint).

Next we contrast that with two quantum circuit training runs but on slightly simpler problems (quantum circuits but still using our classical quantum circuit emulator to compute their result). **Figure 6** shows the quantum training results versus training epoch for a simplified 8-node, 2-truck, including rank 2 only demand structure, and with no cyclic box-return constraint.

Figure 7 shows the quantum training results versus training epoch for the same problem as before, but now with the more realistic rank 2 and rank 3 demand structure. It shows that adding the more realistic rank 3 demand structure significantly hinders the training performance.

Next we compare a simple scenario (similar to Quantum Run 1 in Figure 6) when including the cyclic demand structure or not. First, without the cyclic demand, **Figure 8** shows similar results as before for the quantum training results versus training epoch. And then we see that adding the more realistic cyclic demand structure degrades the training performance. **Figure 9** shows the quantum training results versus training epoch for the same problem as before, but now with the more realistic cyclic rank 2 demand structure.

These results show that the quantum training is challenged to include the more realistic problem features, such as rank 3 demand structure and the cyclic box return constraint. This will limit what we can implement in quantum hardware at this time regarding quantum training.

7.3. Results: AVRP Solutions with Ion Trap Quantum Hardware

And finally we test how well the quantum circuit computes solution inferences using classical, quantum emulator, and quantum hardware (IonQ) solvers in a hybrid scheme. The approach for executing on IonQ is to use a previously trained reinforcement

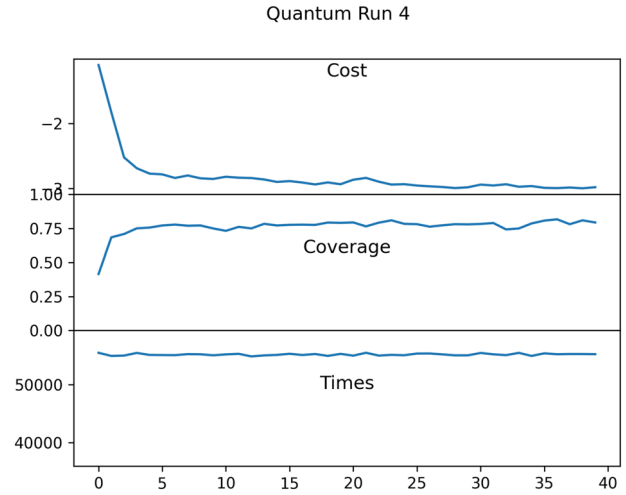


Figure 9. Quantum training run. 2 truck, rank 2 cyclic demand. Uses 8 nodes.

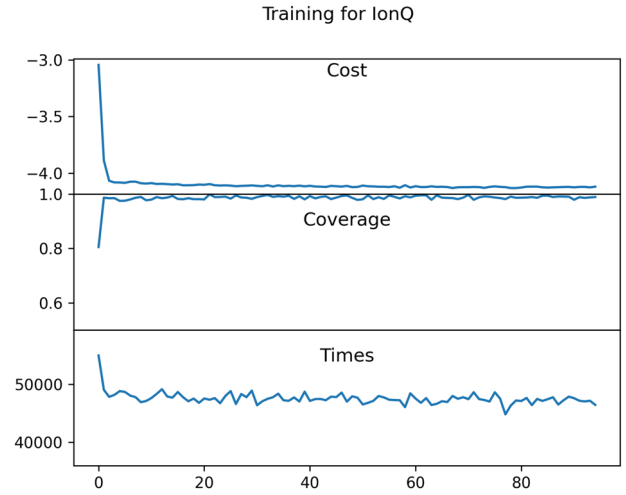


Figure 10. Classical training run. 2 truck, rank 2 only demand, with no cyclic box-return constraint. Uses 8 nodes.

learning agent that was trained classically to capture more of the complexity of the problem (see **Figure 10**). The agent was trained using QC Ware's quantum circuit emulator, but for inference we now use IonQ to compute the quantum layer of the neural network. For every key in two encoder layers, there are eight attention heads. We use an encoding dimension of 64, meaning that each of the eight attention head keys has a dimension of eight itself. We pass each key through an 8 qubit quantum ONN. Moreover, there are eight nodes. This means that we have

$$\underset{\text{nodes}}{8} \times \underset{\text{heads}}{8} \times \underset{\text{encoding layers}}{2} \times \underset{\text{tomography steps}}{3} = 384 \text{ circuits} \qquad (30)$$

Each of these circuits is run with 500 measurements.

7.3.1. Results of IonQ Run

Our two-truck IonQ execution on eight nodes with simplified demand resulted in the following route selection:

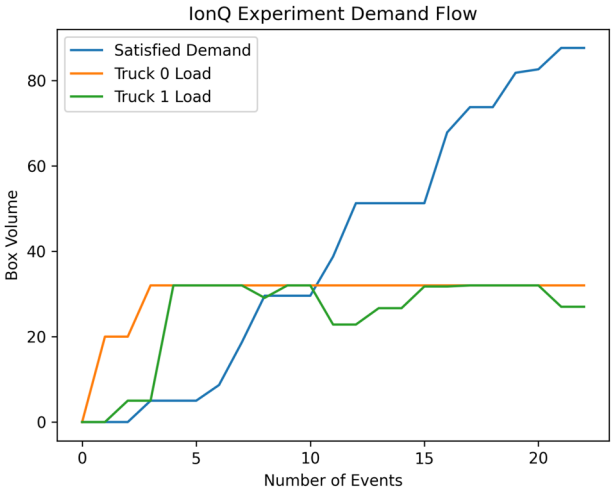


Figure 11. Demand share for IonQ truck assignments versus departure node events. The total initial rank 2 direct demand was 179, so we see that the truck successfully satisfy 49% of demand.

	Truck departure time		Departure node
0	Truck 0	0	Kira plant
1	Truck 1	0	Meiko
2	Truck 0	3960	No.1 and 2 plant
3	Truck 1	6300	Nishio cross-docking
4	Truck 0	7200	Nishio cross-docking
5	Truck 1	11 280	Okazaki East plant
6	Truck 0	15 120	Tahara plant
7	Truck 1	16 380	NO.1 AND 2 plant
8	Truck 1	19 620	Nishio cross-docking
9	Truck 0	23 040	Nishio cross-docking
10	Truck 1	25 860	Meiko
11	Truck 0	28 020	Okazaki East plant
12	Truck 1	31 500	Okazaki East plant
13	Truck 0	33 060	Nishio cross-docking
14	Truck 1	36 540	Nishio cross-docking
15	Truck 0	37 560	Okazaki and electric plant
16	Truck 1	41 040	Okazaki and electric plant
17	Truck 0	42 600	Gamagori plant
18	Truck 1	48 420	Tahara plant
19	Truck 0	49 140	Tahara plant
20	Truck 1	55 020	Gamagori plant
21	Truck 0	55 740	Gamagori plant

The quality of the route found is illustrated by **Figure 11**. This figure shows the volume carried by two trucks at various times associated with departure node events, as well as the total satisfied demand. While the initial demand of 179 is only 49% satisfied, this percentage is actually much better than it sounds. The training data in Figure 10 was done with data that was clipped to very small demand to make training manageable. However, the actual inference run used a much higher demand scale so there is no reason to regard 49% as low coverage.

In principle, we could cover the full Aisin supply chain with multiple teams of two trucks, each computed with IonQ. However, the financial cost of this would be enormous. The run we discuss in this section alone incurred over USD 2000 to run, and this does not account for costs of additional experiments for obtaining improved data. In a related study accomplished by the authors using the same attention head reinforcement learning neural network (but not orthogonal and not quantum) on problems of similar limited size and complexity, the results yielded trained models that are able to satisfy about 75% of the full daily demand met by logistics experts using heuristics and spreadsheets. What would be needed to improve overall performance of the QONN is to use more trucks, the full 2 and 3 rank demand in training, and larger quantum layers. This will be demanding for classical and emulator versions of the QONN and cost prohibitive for runs on quantum hardware.

A key part of future studies on the QONN approach will be the ability of future quantum hardware to compute larger circuits with less error. The benchmarking method in Subsection 7.1 will be critical in determining how many qubits and how deep the circuits can be to still give reliable measurements of the quantum layers of the QONN.

7.3.2. Scaling of IonQ Run

In discussing the scaling of the QONN for AVRP solutions there are two aspects to consider: 1) the scaling of the quantum circuit layer solutions and 2) the scaling of the overall reinforcement learning algorithm.

First, let us consider the scaling of the quantum circuit measurements. In theory, the more nodes (and qubits) in the quantum layer of the neural network the better in terms of providing computational benefit, but this is difficult to quantify. As the results in Figures 1–4 show, the performance of the circuit is quite sensitive to the noise and imprecision inherent in today's state of the art quantum processor technology. Again, qualitatively, the 6-qubit results look good, the 10-qubit results look bad, and the 8-qubit results look questionable. But after a bit of empirical testing in the QONN training, we decided to execute the full experiment using an 8-qubit quantum layer. This gave us a little bit more quantum power at the cost of additional noise. In this application, the QONN is slightly forgiving in slight computational variations and in some sense these even help prevent over fitting. However, we hesitate to make a virtue out of a vice.

As quantum hardware improves, practitioners will always seek to get the most performance as possible out of the processor and will always be empirically pushing against the wall of degraded results from accumulated error, at least until we reach an error-corrected level of maturity. For this study, comparing AVRP runs using different numbers of qubits in the quantum layer would be dominated by noise in the circuit. An interesting study could be done in the future when quantum hardware has improved, for example, to compare a reliably computed 16-qubit quantum layer to a reliably computed 8-qubit quantum layer in our QONN.

Second, what is the scaling behavior of the trained agent from the neural network itself, whether trained classically or quantumly? This can be an open question in terms of quality of solution desired and computational resources available. However, this aspect we have not fully studied in this and related work .[3] Much additional work could be done in improving the classical version of this ONN—larger neural network architectures, larger attention head layers, better tuned meta-parameters, and more.

The key question comes in combining the two. For example, improving the classical structure of the ONN and then adding the quantum layer, and running the experiment for two different sizes of quantum layer, such as 8 qubits and 16 qubits discussed above. That would allow a much better quantitative assessment of the benefit versus cost of the quantum layer in the overall AVRP solution.

And finally, we must observe that we are still a long way from solving truly large problems of industrial scale. Training teams of two or three trucks on 21 nodes with training times on the order of 10 min (classical and emulated) are still relatively small machine learning problems. These small problems parameters were chosen as they enabled runs on the quantum hardware that were reliable and affordable. How long it will take for quantum hardware to mature to handle much larger problems is unknown. However, we find this experiment to be one of the largest industrial problems to have been addressed by quantum hardware, albeit in a classical-quantum hybrid approach.

8. Conclusions

While there is much additional work to do in exploring algorithmic and workflow improvements, these first results demonstrate that quantum circuits can successfully train a reinforcement model comprised of ONNs to solve the logistics problem. While the quantum versions of ONNs, QONNs, have theoretically provable speedups, it remains to be seen how much of that theoretical advantage can be realized in practice.

To that end, this work demonstrated the successful implementation of QONNs applied to a practical problem of real world importance and developed several methods of tuning and measuring performance, both of the quantum circuit primitives used in the QONN and in the overall meta-parameters used in constructing the training and inference workflows.

We were only able to study a few approaches of implementing the QONN, quantum circuits, attention heads, layering depth, demand structure, and constraints during the course of this project. Further work would likely find more efficient circuits and QONN implementations along with more effective training approaches.

We emphasize that the errors in quantum computing we are seeing here are not surprising and reflect the early state of quantum hardware. So it will be important in the future to continue monitoring the improvements in quantum hardware and the performance on algorithmic primitives to discern when they might improve computational workflows.

Conflict of Interest

The authors declare no competing non-financial interests. The authors do disclose the following financial interests. S.W., F.S., and R.C. were all employed by QC Ware during all or portions of this research. R.C. is a co-founder of QC Ware Corp. S.W., F.S., and R.C. hold QC Ware stock or stock options. T.I. and K.K. were employed by Aisin Group. Aisin Group is a customer of QC Ware; Aisin Group provided funding for the research presented in this manuscript.

Data Availability Statement

The only data used in this research were the details of the logistics nodes, trucks, and parts demand, which are withheld for proprietary commercial reasons. The quantum circuit used in the study and the results of the reinforcement learning are presented in the paper.

Keywords

hybrid quantum classical algorithm, quantum circuit, quantum orthogonal neural network, quantum tomography, trapped ion qubits

Received: December 4, 2022
Revised: March 14, 2023
Published online: May 18, 2023

[1] G. B. Dantzig, J. H. Ramser, *Manage. Sci.* **1959**, *6*, 80.
[2] P. Toth, D. Vigo, *The Vehicle Routing Problem*, SIAM, Philadelphia **2002**.
[3] R. Correll, S. Weinberg, F. Sanches, T. Ide, T. Suzuki, *arXiv:2211.17078*, **2022**.
[4] W. Kool, H. Van Hoof, M. Welling, *arXiv:1803.08475*, **2018**.
[5] I. Bello, H. Pham, Q. V. Le, M. Norouzi, S. Bengio, *arXiv:1611.09940*, **2016**.
[6] S. Lloyd, M. Schuld, A. Ijaz, J. Izaac, N. Killoran, *arXiv:2001.03622*, **2020**.
[7] I. Gianani, I. Mastroserio, L. Buffoni, N. Bruno, L. Donati, V. Cimini, M. Barbieri, F. S. Cataliotti, F. Caruso, *Adv. Quantum Technol.* **2022**, *5*, 2100140.
[8] M. Kordzanganeh, P. Sekatski, L. Fedichkin, A. Melnikov, arXiv:2212.00736, **2022**.
[9] E. Grant, L. Wossnig, M. Ostaszewski, M. Benedetti, *Quantum* **2019**, *3*, 214.
[10] A. Skolik, J. R. McClean, M. Mohseni, P. vander Smagt, M. Leib, *Quantum Mach. Intell.* **2021**, *3*, 5.
[11] J. Alcazar, V. Leyton-Ortega, A. Perdomo-Ortiz, *Mach. Learn.: Sci. Technol.* **2020**, *1*, 035003.
[12] K. Batra, K. M. Zorn, D. H. Foil, E. Minerali, V. O. Gawriljuk, T. R. Lane, S. Ekins, *J. Chem. Inf. Model.* **2021**, *61*, 2641.
[13] A. Sagingalieva, M. Kordzanganeh, N. Kenbayev, D. Kosichkina, T. Tomashuk, A. Melnikov, arXiv:2211.05777, **2022**.
[14] A. Sagingalieva, A. Kurkin, A. Melnikov, D. Kuhmistrov, M. Perelshtein, A. Melnikov, A. Skolik, D. Von Dollen, arXiv:2205.04878, **2022**.
[15] J. Landman, N. Mathur, Y. Y. Li, M. Strahm, S. Kazdaghli, A. Prakash, I. Kerenidis, *Quantum* **2022**, *6*, 881.
[16] S. Li, K. Jia, Y. Wen, T. Liu, D. Tao, *IEEE Trans. Pattern Anal. Mach. Intell.* **2019**, *43*, 1352.
[17] I. Kerenidis, J. Landman, N. Mathur, *arXiv:2106.07198*, **2021**.
[18] S. Weinberg, F. Sanches, T. Ide, K. Kamiya, R. Correll, *Sci. Rep.* **2023**, *13*, 4770.
[19] F. Sanches, S. Weinberg, T. Ide, K. Kamiya, *Phys. Rev. A* **2022**, *105*, 062403.
[20] R. J. Williams, *Mach. Learn.* **1992**, *8*, 229.
[21] S. Ioffe, C. Szegedy, in *Proc. of International Conference on Machine Learning*, MLR Press, New York **2015**, pp. 448–456.
[22] R. S. Sutton, A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed., The MIT Press, Cambridge, MA **2018**.
[23] K. Wright, K. M. Beck, S. Debnath, J. M. Amini, Y. Nam, N. Grzesiak, J. -S. Chen, N. C. Pisenti, M. Chmielewski, C. Collins, K. M. Hudek, J. Mizrahi, J. D. Wong-Campos, S. Allen, J. Apisdorf, P. Solomon, M. Williams, A. M. Ducore, A. Blinov, S. M. Kreikemeier, V. Chaplin, M. Keesan, C. Monroe, J. Kim, *Nat. Commun.* **2019**, *10*, 5464.