A Hybrid Quantum-Inspired and Deep Learning Approach for the Capacitated Vehicle Routing Problem with Time Windows

Jorin Dornemann[1][0000−0003−3518−6039], Salwa Shaglel[2][0000−0002−4520−0167], Martin Kliesch[2][0000−0002−8009−0549], and Anusch Taraz[1][0000−0003−3646−3683]

[1] Institute of Mathematics, Hamburg University of Technology, 21073 Hamburg, Germany jorin.dornemann@tuhh.de
[2] Institute for Quantum Inspired and Quantum Optimization, Hamburg University of Technology, 21073 Hamburg, Germany

Abstract. This paper introduces a hybrid approach to address the Capacitated Vehicle Routing Problem with Time Windows by integrating quadratic unconstrained binary optimization (QUBO) hardware with deep learning-assisted heuristics. The proposed three-phase heuristic leverages the strengths of QUBO-solving hardware while mitigating its limitations, aiming at offering better scalability to larger problem instances. In the first phase, a deep learning-enhanced QUBO formulation is employed to partition the vertices into clusters. The second phase uses deep learning-assisted tree searches to generate candidate routes within each cluster. These candidate routes are combined in the third phase into a feasible global solution by solving a quadratic unconstrained binary set partition problem. This framework ensures compliance with capacity and time window constraints while maintaining computational efficiency. Computational results indicate that the hybrid approach is promising to potentially scale well for larger problem cases while respecting hardware limitations, offering a viable approach for leveraging quantum-inspired hardware in combination with advanced heuristics for solving complex combinatorial optimization problems.

Keywords: Vehicle Routing · Deep Learning · Quantum-inspired Computing

1 Introduction

The capacitated vehicle routing problem with time windows (CVRPTW) is an extension of the travelling salesman problem (TSP), where the goal is to find a set of routes, such that all vertices except for a depot node are visited exactly once and the capacity and time window constraints are met. The objective is to find routes to minimize the total distance traveled. First introduced by Solomon [25], the problem has been extensively studied since and has found application in various practical scenarios. Given a fixed number of vehicles, even finding a feasible solution is NP-hard [4]. Therefore, various approaches have been proposed to address the CVRPTW over the years. Recent developments in both

software and hardware have inspired novel approaches to address this problem. Advancements in machine learning have provided valuable tools for extracting essential graph properties in routing problems [27], [13]. Moreover, specialized hardware is being developed to address combinatorial optimization problems, including quantum computers. Oftentimes, quantum and quantum-inspired algorithms and devices are designed to solve quadratic unconstrained binary optimization (QUBO) problems. Many combinatorial optimization problems can be formulated as QUBOs [17], hence, it is increasingly intriguing to explore ways to utilize specialized hardware for solving combinatorial optimization problems formulated as QUBOs. However, since these hardware systems can handle only a limited number of variables so far, it is essential to develop new scalable methods for integrating this hardware into a solving framework.

In this work, we propose a hybrid approach to approximately solve the CVRPTW. The proposed 3-phase heuristic combines quadratic unconstrained binary optimization hardware with a deep learning-assisted heuristic. The goal is to leverage the strengths of QUBO-solving hardware while ensuring that the hardware limits are not exceeded. For this purpose, we first use a deep learning-complemented QUBO formulation to cluster the set of vertices into subsets, generate sets of candidate routes for each cluster using deep learning-assisted tree searches that are then combined into a complete solution by a quadratic unconstrained binary set partition problem.

We solve the QUBO problems using Fujitsu's Digital Annealer (DA) [18] in its third generation (v3). The DA uses classical dedicated hardware with the purpose to heuristically solve QUBO problems as fast as practically possible. Ultimately, the goal is to use quantum computers for this task and the DA is often considered to be an intermediate quantum-inspired solution that allows to assess some of the potential of quantum computers already today. Technically, the DAv3 uses an application-specific integrated circuit (ASIC) implementing a Markov Chain Monte Carlo (MCMC) method. This method is then called by a global tabu search that can handle QUBO instances up to 100,000 bits. A major advantage of the DA is its massively parallel implementation and an innovative MCMC sampling technique. In comparison to standard simulated annealing, its update steps typically have a much smaller rejection rate speeding up the optimization, and one update step is made within one clock cycle of the ASIC. The DA also supports parallel tempering, which improves the dynamic properties of the Monte Carlo method. Moreover, it uses dynamic off-setting of the objective function to escape local minima in order to explore the optimization landscape more comprehensively. This explains why the DA can outperform other known approaches to solving QUBO problems [15].

2 Related work

For the CVRPTW, there exist only a limited number of approaches that cluster the set of customers while taking into account the time window constraints. Most methods focus on the spatial characteristics of the customer's features.

The sweep-based heuristic [9], [25] clusters the customers based on their polar coordinates based on a center of gravity, which in the context of vehicle routing is the depot. An imaginary ray originating from the depot is swept counter-clockwise over all vertices. The demand of each swept vertex is accumulated. Once the accumulated demands reach the capacity for one vehicle, or if including the next vertex exceeds the capacity, the current nodes are included in a cluster. Other spatial approaches include [19], [1] and [5]. Qi et al. [22] use time geography theory to represent time and space in the same coordination system defining a spatiotemporal distance, which is used to build clusters minimizing this distance. To our knowledge, the only literature that utilizes machine learning assistance within a clustering algorithm for the CVRPTW is the one by Poullet [21]. They develop an optimal classification tree method [2] to predict the number of vehicles.

The utilization of specialized QUBO hardware for solving vehicle routing and similar problems is still in its early stages of exploration. Most of these approaches have a hybrid structure, as the physical limitations of specialized hardware require dividing the problem into smaller subproblems that can be handled. Tran et al. [26] propose a hybrid quantum-classical tree search, where a classical processor maintains a global search tree and enforces constraints on relaxed sub-problems, and a quantum annealer is applied to obtain strong candidate solutions by sampling from the configuration space of the relaxed problem. Rieffel et al. [23] investigate the effectiveness of a quantum annealer in solving small instances within families of hard operational planning problems. Feld et al. [8] follow the *Cluster First, Route Second* [16] approach to solve the capacitated vehicle routing problem (CVRP) using D-Waves quantum annealer. In their 2-Phase heuristic they divide the set of customers into clusters by formulating this as a QUBO knapsack problem with additional distance minimization. Each cluster is then mapped to a TSP QUBO formulation and subsequently solved using the quantum annealer. Irie et al. [12] develop a QUBO formulation for the CVRP that incorporates constraints with time, state and capacity and conduct experiments on D-Waves quantum annealer, but their formulation quickly exceeds the hardware limits even for smaller instances.

3 Three-Phase Heuristic

Solving the CVRPTW formulated as a QUBO directly using quantum(-inspired) computing is currently not a feasible option because the number of binary variables required to represent an instance quickly surpasses the capabilities of dedicated hardware like the DA. Therefore, we propose an extended version of the *Cluster First, Route Second* method. The three phases are:

1. **Split** the set of vertices into a number of k clusters by solving a cluster optimization problem, complemented with edge weights given by a deep neural network, formulated as a quadratic unconstrained binary optimization problem on the DA. Repeat this for different values of k to diversify the set of clusters.

2. **Generate** a set of different feasible candidate routes for the CVRPTW on each of the found clusters by applying a deep learning-assisted tree search heuristic.
3. **Solve** a set partition problem (SPP) formulated as a QUBO on the DA to compose a complete solution by choosing a subset of the set of candidate routes generated in Step 2 minimizing the total costs.

This approach offers the advantage of significantly reducing the size of the QUBO matrix solved by the DA compared to solving a CVRPTW instance directly on it. The heuristic applied in step 2 is highly flexible, as it does not attempt to solve each cluster as a single-route instance but instead finds the best set of routes within each cluster and can build different sets of routes for each cluster in parallel. This flexibility allows us to vary the number of clusters in step 1 by a parameter λ that we call *cluster range*, forcing different cluster assignments and thereby creating a more diverse set of candidate routes for inclusion in step 3. In this way, we leverage the strengths of the DA, particularly its ability to handle binary quadratic unconstrained optimization problems with a manageable number of binary variables and delegate the handling of the hard capacity and time window constraints to a heuristic for more efficient processing.

3.1 Problem Definition

A CVRPTW instance is given as a directed complete graph $G = (V \cup \{0\}, E)$ with $n+1$ nodes, where node 0 is the special depot node. Additional constraints are imposed by a demand and time window attached to each node i, given as d_i and $[a_i, b_i]$, respectively, with $[a_0, b_0]$ representing the planning horizon regarding the earliest possible departure from and latest possible return to the depot. Each vehicle has a capacity of Z, which the sum of the demands along the vehicle's route must not exceed. The provided time window $[a_i, b_i]$ at each node $i \in V$ represents the time interval in which the service at node i is allowed to start. An arrival at node i before a_i is allowed, the vehicle then has to wait until a_i to start the service. Furthermore, each node $i \in V$ requires a specific service duration h_i. We assume a homogeneous fleet of vehicles so that the capacity and travel time are equal for all vehicles. Furthermore, the edge weights c_{ij} for each edge $e = (i, j)$ represent the transit costs from node i to node j and without loss of generality include the service duration h_i of node i. The objective is to minimize the total distance traveled over all vehicles.

3.2 Graph Convolutional Network

For our approach, we use a Residual Gated Graph Convolutional Neural Network (GCN) [3], [13], which is adapted for the CVRPTW in [6] by modifying the layers to account for additional constraints and briefly described in the following.

The neural network assigns a value to each edge in the graph to predict which edges are most promising to include in a solution. These associated edge values can be interpreted as probabilities. In the following, we describe each of the neural network's layers.

Input Layer The input for the node features is five-dimensional. For node i we have the two-dimensional coordinates $x_i \in [0,1]^2$, the time window given as $[a_i, b_i]$ and the normalized demand d_i/Z, where we set $d_0 = 0$ for the depot. These features are concatenated to the five-dimensional input feature vector y_i and are then embedded to an $\frac{h}{2}$-dimensional representation, where h denotes the hidden dimension of our network that represents the number of parameters each hidden state within the network stores. The depot node gets a separate learned initial embedding parameter in order to mark the depot node as special for the network. For that, define $\hat{y}_0 \in \{0,1\}^{n+1}$ to be the unit vector with entry one at the first position and zeros otherwise. This is put together as the node input feature as follows:

$$\alpha_i = A_1 y_i \oplus A_2 \hat{y}_0, \tag{3.1}$$

where $A_1 \in \mathbb{R}^{\frac{h}{2} \times 5}$, $A_2 \in \mathbb{R}^{\frac{h}{2} \times (n+1)}$ are the weight parameters that are learned during the training procedure and $\cdot \oplus \cdot$ is the concatenation operator.

For the input edge feature, the edge values c_{ij} are embedded as an $\frac{h}{2}$-dimensional feature vector. We use an indicator function δ_{ij} of an edge which has the value one for edges connecting nodes i and j, with $i \neq j$ and i, j not the depot, and value two for edges connecting nodes with itself. To tag the depot as a special node, the indicator function δ_{ij} furthermore has a value of 3 for edges to and from the depot and a value of 4 for the depot self-loop. Together, the edge input feature is given as:

$$\beta_{ij} = A_3 c_{ij} \oplus A_4 \delta_{ij}, \tag{3.2}$$

with $A_3 \in \mathbb{R}^{\frac{h}{2} \times 1}$ and $A_4 \in \mathbb{R}^{\frac{h}{2} \times 1}$as above are weight parameters to be learned. These $\frac{h}{2}$-dimensional edge and node feature representations are then concatenated to form the h-dimensional input for the first graph convolution layer and are subsequently passed through all graph convolution layers before producing our desired output within our classifier layers.

Graph Convolution Layer In each of the graph convolution layers the model updates the edge and node embeddings. We leverage the design of the residual gated graph convolutional network developed in Bresson et al. [3] by adding an edge feature representation. Let ℓ be the current layer and for node i and edge (i,j), let x_i^ℓ be the node features vector and e_{ij}^ℓ the edge features vector. We define the features for layer $\ell + 1$ in the following way:

$$x_i^{\ell+1} = x_i^\ell + \mathrm{ReLU}\left(\mathrm{BN}\left(W_1^\ell x_i^\ell + \sum_{j \in N(i)} \eta_{ij}^\ell \odot W_2^\ell x_j^\ell\right)\right), \tag{3.3}$$

$$e_{ij}^{\ell+1} = e_{ij}^\ell + \mathrm{ReLU}\left(\mathrm{BN}\left(W_3^\ell e_{ij}^\ell + W_4^\ell x_i^\ell + W_5^\ell x_j^\ell\right)\right), \tag{3.4}$$

where $W_k^\ell \in \mathbb{R}^{h \times h}$ for $k \in [5]$ again are the weight parameters to learn, with the notation $[n] = \{1, \ldots, n\}$ for any positive integer n, ReLU being the rectified linear unit, BN stands for batch normalization, $N(i)$ denotes the neighborhood

of node i, moreover $\odot$ denotes the Hadamard product operator and η_{ij}^{ℓ} being defined as $\eta_{ij}^{\ell} = \frac{\sigma(e_{ij}^{\ell})}{\sum_{j' \in N(i)} \sigma(e_{ij'}^{\ell}) + \varepsilon}$ with σ being the sigmoid function and ε a small value. For the input layer, we set $x_i^0 = \alpha_i$ and $e_{ij}^0 = \beta_{ij}$. We implement W_5^{ℓ} as a separate parameter to allow the model to distinguish different directions of edges since in the context of CVRPTW we have directed edges in our solutions.

MLP Classifier A multi-layer perceptron (MLP), which is a fully connected feedforward neural network with a number ℓ_M of hidden layers, is used for generating the desired output, a finite measure that represents probabilities over the edges of our fully connected graph. For each edge embedding e_{ij}^L of the last graph convolution layer L, the MLP outputs the *probability* p_{ij} that this edge is included in the tours of the CVRPTW solution: $p_{ij} = \mathrm{MLP}(e_{ij}^L)$. The edge representations are linked to the ground-truth tour through a softmax output layer, which allows us to train the model parameters end-to-end by minimizing the cross-entropy loss via gradient descent.

3.3 Vertex Clustering

In each iteration of step 1, we split the set of vertices into a varying number of k subsets to vary the assignment of vertices to the clusters and subsequently obtain a more diverse set of candidate routes in step 2. To achieve this, we bound k from above by the minimal number of vehicles needed to solve the instance, as a number of clusters exceeding this bound could potentially exclude the overall optimal solution to this instance by forcing more routes than necessary. The bound $\bar{k}$ is given as the sum over all demands divided by the capacity of a vehicle. We define our decision variables x_{vm} to be

$$x_{vm} = \begin{cases} 1 & \text{if vertex } v \text{ is assigned to cluster } m, \\ 0 & \text{otherwise.} \end{cases} \tag{3.5}$$

With this notation, we can formulate the clustering problem as a quadratic integer program as follows:

$$\begin{aligned} \min \quad & \sum_{m=1}^{k} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{im} x_{jm} \quad \text{s.t.} \\ & \sum_{m=1}^{k} x_{im} = 1 \quad \forall i \in V, \\ & x_{im} \in \{0,1\} \quad \forall i \in V, m \in [k], \end{aligned} \tag{3.6}$$

where we minimize the sum of costs between all vertex pairs in each cluster under the constraint that each vertex belongs to exactly one cluster. The equivalent

QUBO formulation is given as:

$$\min \quad \sum_{m=1}^{k}\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{im}x_{jm} + P\left(\sum_{i=1}^{n}\left(\sum_{m=1}^{k} x_{im} - 1\right)^2\right), \tag{3.7}$$

where the constraint is added as a penalty term with a penalty factor $P \geq 0$. This QUBO problem is subsequently solved using the DA [18]. In contrast to other formulations, we do not include inequality constraints ensuring that the sum of the demands of each cluster does not exceed the capacity of the vehicles as well as taking into account the time windows within the clusters, as we do not solve each cluster with one route. This aligns with the strengths of the DA, as inequalities and integer variables cause difficulties.

By augmenting the objective function with additional terms, we can incorporate additional information gained by the deep neural network (cf. Section 3.2) into the cluster decision process instead of relying solely on the costs c_{ij}. For this purpose, let p_{ij} be the probability to be included in the correct solution for each edge (i, j), assigned by the neural network. Note that this associated probability value is directional, i.e., $p_{ij} \neq p_{ji}$. For the clustering procedure, however, the directions are not relevant and would even distort the result, as a large probability for one direction does not mean that the probability for the other direction is also large. We therefore take the maximum value of the two directions: $\overline{p_{ij}} = \max\{p_{ij}, p_{ji}\}$. A shortcoming of using these probabilities straightforwardly for clustering is that it may overlook important relationships between nodes. If we have edges (v_1, v_2) and (v_2, v_3) with high probabilities, but the neural network assigns a low probability to the edge (v_1, v_3) because it recognizes the benefit of visiting v_2 between v_1 and v_3, we want to incorporate the insight that v_1 and v_3 should be assigned to the same cluster into the optimization problem. We do this by replacing the probabilities p_{ij} in (3.9) by *path probabilities*, which we define as:

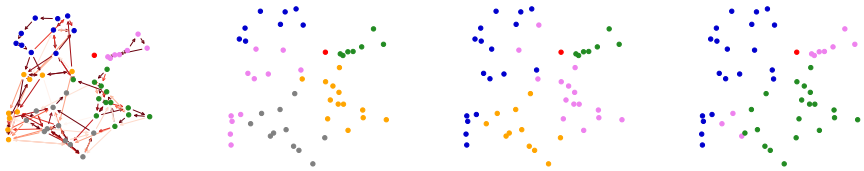
$$p_{ij}^{\text{path}} := \max\left\{\prod_{(v,w)\in P} p_{vw} : P \text{ is a path from } i \text{ to } j\right\}. \tag{3.8}$$

Then the clustering problem can be formulated as follows:

$$\begin{aligned}
\min \quad & \alpha_{\text{dist}} \sum_{m=1}^{k}\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{im}x_{jm} \\
& + \alpha_{\text{prob}} \sum_{m=1}^{k}\sum_{i=1}^{n}\sum_{j=i+1}^{n} (1 - \overline{p_{ij}^{\text{path}}})x_{im}x_{jm} \quad \text{s.t.} \\
& \sum_{m=1}^{k} x_{im} = 1 \quad \forall i \in V, \\
& x_{im} \in \{0, 1\} \quad \forall i \in V, m \in [k],
\end{aligned} \tag{3.9}$$

where α_{dist} and α_{prob} denote the tuning parameters assigned to each part of the objective function to weigh the different arguments.

Example 1. Figure 1 shows an example solution of the clustering problem (3.9) for an instance with 50 nodes. Figure 1(a) shows the path probability heatmap, where only edges with an associated value greater than 0.25 are displayed, where a darker red indicates a higher value. Figure 1(b)-(d) show the clusters built by solving (3.9), where each cluster is depicted as a different node color, for the different values of $k = 3, 4, 5$.



(a) Heatmap (b) $k = 5$ (c) $k = 4$ (d) $k = 3$

Fig. 1: Path probability heatmap and solutions for clustering problem (3.7) for different numbers k of clusters

3.4 Candidate Route Generation

Each cluster $C_i, i \in [k]$, is solved as a CVRPTW instance. Note that a CVRPTW solution can consist of several routes. Since the purpose of this step is to generate candidate routes to be included in the SPP in step 3, which then selects routes from the set of generated candidate routes in step 2 to form a feasible solution to the complete instance, it is beneficial to not only create one set of candidate routes per cluster.

For this, we adapt a deep neural network-assisted beam search [6] that utilizes the GCN presented in Section 3.2. The beam search is designed to build a number of solutions in parallel. In [6] it is shown that this heuristic works quite well for smaller instances. In the context of our 3-phase heuristic we use the beam search to generate a set candidate routes by considering all individual routes of all built solutions by the beam search, instead of choosing one solution built by the beam search and disregarding all others, as done in [6]. Furthermore, by applying the beam search only to the smaller clusters generated in step (1), we mitigate the inaccuracies of the beam search for larger instances, as shown in [6].

A beam search with beam width b is a limited-width breadth-first search, such that b sets of routes are built simultaneously. It builds a search tree iteratively, where each tree node represents a partially constructed solution. This heuristic uses the values p_{ij} associated with each edge $e = (i, j)$ of the graph, as described in Section 3.2, to build solutions that are most promising according to the values p_{ij}.

Starting from the root node that holds the partial solution [0], i.e., being at the depot, in each layer of the beam search tree, only the b most promising partial solutions with respect to a scoring policy S are further explored. The descendants of each search tree node are those that add a vertex to the partial solution of its parent that is compliant with the side constraints of the CVRPTW. The scoring policy S of a (partially) built solution $g = [v_1 \ldots, v_m]$ is given as

$$S(g) = \prod_{i=1}^{m-1} p_{v_i v_{i+1}},$$

where a partial solution g can be interpreted as an array starting at the depot node 0 and then following a trail in the graph where the only node to be included more than once in g is the depot node.

Let $\tilde{n}$ be the number of vertices in one of the clusters including the depot for which we want to build a set of solutions by applying the beam search. Let $S^\ell \in \mathbb{R}^{b \times \tilde{n}}$ be the scoring matrix at the $\ell-$th iteration of the beam search, which is subsequently calculated in each iteration of the beam search, and $g_{b'} = [v_1, \ldots, v_\ell]$ be one of the b partially built solutions after ℓ iterations. The $w-$th entry in the $b'-$th row of the scoring matrix $S^{\ell+1}$ then reflects the score if the partial solution $g_{b'}$ is continued by adding the vertex w. That is, $S^{\ell+1}$ is given as the policy score $S(g_{b'})$, multiplied by the probability given by the neural network of getting from v_ℓ to w in the next step for all nodes $w \in [\tilde{n}]$:

$$S^{\ell+1}_{b'w} = S(g_{b'}) \cdot p_{v_\ell w}.$$

Then, the invalid expansions, including the already visited nodes, are excluded within the scoring matrix $S^{\ell+1}$ by masking the values that correspond to infeasible continuations of the partial solutions to be zero. From the $b \cdot \tilde{n}$ values in the masked scoring matrix the b highest scores are chosen, and the b partial solutions represented by these b search tree nodes are continued while the other are discarded. The beam search stops when b complete solutions are built.

3.5 Set Partition Problem

As described in [4], the CVRPTW can be formulated as a set partition problem (SPP), which we will define in the following. The SPP is the basis of the currently best exact method for the CVRPTW, the branch-cut-and-price algorithm [4]. It uses the SPP as the master problem, as the solution of the linear relaxations of the SPP provides better lower bounds than, for example, the linear relaxation of the mixed integer linear program (MILP) formulation of the CVRPTW.

For this purpose, let R be the set of all feasible routes on subsets of vertices. For $r \in R$ denote with c_r the cost of route r, let δ_{vr} be a binary coefficient that is equal to one if and only if vertex $v \in V$ is in route r. Define the decision variable y_r to be

$$y_r = \begin{cases} 1 & \text{if route } r \text{ is used in the solution,} \\ 0 & \text{otherwise.} \end{cases} \tag{3.10}$$

Then the set partition problem can be formulated as an integer program

$$\begin{aligned} \min \quad & \sum_{r \in R} c_r y_r \quad \text{s.t.} \\ & \sum_{r \in R} \delta_{vr} y_r = 1, \quad v \in V \\ & y_r \in \{0,1\} \quad \forall r \in R, \end{aligned} \tag{3.11}$$

where we minimize the costs of the chosen subsets of routes under the constraint that each vertex appears in exactly one route. For a given solution y of this problem we can link together all routes indicated by y to obtain a feasible solution to the instance with respect to the side constraints such that every node is visited and every route starts and ends at the depot. This integer program translates to the following QUBO formulation:

$$\min \quad \sum_{r \in R} c_r y_r + P \left(\sum_{v \in V} \left(\sum_{r \in R} \delta_{vr} y_r - 1 \right)^2 \right), \tag{3.12}$$

with P being a penalty factor. This QUBO problem is then solved by the DA. With R being the set of all feasible routes, this corresponds to the optimal solution of the CVRPTW. But in general, finding the optimal solution for this problem is only practically possible for small instances if all feasible routes are included in R, as the number of routes grows exponentially fast. For our approach, we do not choose R as the set of all possible routes but only include the subset of routes generated as described in Section 3.4 in the consideration, such that the number of routes included in the SPP is easily controllable via the beam width b.

4 Computational Experiments

This section evaluates the overall performance of our approach by comparing its results to those of other methods, including state-of-the-art solutions for the CVRPTW.

All models are implemented in Python 3.10 and run under Windows 10. The neural network architecture is implemented using PyTorch version 2.0.0 to use GPU computation with Cuda version 11.8. The network consists of $\ell_{\text{GCN}} = 30$ hidden layers and $\ell_M = 3$ layers in the MLP. We use a hidden dimension $h = 300$ in each of the layers. For each problem size, an individual neural network is trained on 1 million randomly sampled instances based on the distribution given in the R201 instance of [25], which consists of randomly generated geographical data, a long scheduling horizon and short to medium-sized time windows allowing only a few customers per route. The instances were solved to optimality by Gurobi [10] for 20 node instances, whereas the instances with 50 and 100 customers, respectively, were solved using one run of LKH [11] and are therefore not necessarily optimal. We apply a supervised learning procedure, where, given as

input a graph with the additional node features time windows and demands, the model is trained to output a probability matrix by minimizing the cross-entropy loss via gradient descent with respect to the adjacency matrix corresponding to the target solution. We utilize the Adam optimizer [14] along with a gradual decrease in the learning rate for smoother convergence, starting at a rate of 10^{-3}. We train the nets using a batch size of 24 for 1500 epochs with 500 randomly chosen batches and select the point of training with the lowest validation loss. The training procedure is executed on machines with two CPUs of type Intel Xeon E5-2680v3 @ 2,50GHz with 12 Cores and four NVidia Tesla K80 GPUs with 12GB RAM.

The path probabilities p_{ij}^{path} are calculated by applying the Floyd-Warshall algorithm to the graph (G, ω) with the weight function $\omega : E \mapsto \mathbb{R},\ \omega((i,j)) = -\log(p_{ij})$ to find shortest paths. With the weight function ω defined like this, a shortest path in (G, ω) corresponds to maximizing (3.8).

To find the best configuration of weights α_{dist} and α_{prob} in the objective function (3.9), each combination of weights in $\{0.0, 0.1, \ldots, 0.9, 1.0\}$ summing up to one is tested for combinations of the distance c, the edge probabilities p given by the neural network and the path probabilities p^{path}. The best weighting of the objective function given by the manual experiments is given as $\alpha_{\text{dist}} = 0.2, \alpha_{\text{prob}} = 0.8$ using the distance and path probabilities. The accuracy of determining clusters based solely on distance differences is lower because the time window constraints in CVRPTW significantly impact route feasibility, making spatial arguments less relevant.

To set the beam width as well as the time limit for calculations using the DA, we manually conduct experiments on small sets of instances for all problem sizes to test different impacts. First, given the clusters that correspond to the optimal routes in the best solution, we test the impact of beam width and time limit for the SPP optimization problem. Second, given the clusters found by solving (3.7) with the DA and the routes built by the beam search for each cluster, we include the routes of the correct solution to find the average calculation time it takes the DA to find this solution when solving the SPP (3.12). Based on these experiments, for the beam width, we choose $b = 50, 150$ and 300 for $n = 20, 50, 100$, respectively. For $n = 20$, we choose a cluster range $\lambda = 2$, for the other problem sizes $\lambda = 3$. Even one to two seconds is enough for the DA to solve the SPP (3.12) to optimality for 20 node instances. To account for the increase of variables when solving (3.7) for multiple values of k, we choose a time limit of 5, 15 and 50 seconds for $n = 20, 50, 100$, respectively, for the DA to solve the SPP.

In Table 1, we compare our approach to the commercial exact MILP solver Gurobi version 10.0.0 [10] as well as to the best known exact solution method for vehicle routing problems, branch-cut-and-price (BCP) algorithm [24], [7]. We further compare results to the heuristic LKH3 [11] and Google's OR-Tools (GORT) version 9.5.2237 library [20], both of which frequently serve as heuristic baselines in related literature. The comparison is done by calculating the percentage *gap* of the cost of the found solution to the cost of the best known solution for each

instance. The runtime for our approach is measured as the time for the execution of all three steps without the communication and idle time with the DA. The values are averages over 100 instances for each problem size n.

For GORT, one can choose different configurations regarding the underlying local search heuristic. We chose guided local search (GLS) to be best suited for our needs. As a time limit is needed for the GLS, we have chosen time limits of 10, 100, and 300 seconds for the different problem sizes. These time limits are selected to align with the runtime of our method, which allows for a fair comparison between the approaches. Gurobi is applied to the 2-index Integer Linear Program formulation of the CVRPTW and is given a time limit of 10, 100 and 300 seconds per instance for the different problem sizes, respectively, to allow for a comparison with similar running times. We evaluate the baseline models on machines with two Intel Xeon E5-2680v3 @ 2.50GHz CPUs with 12 cores and 128GB RAM available at the High Performance Computing Cluster of Hamburg University of Technology, while the execution of our 3-phase heuristic is done on a local machine with an Intel Core i7-1165G7 CPU @ 2.80GHz and 16GB RAM running on Windows 11. The reason for this is that communication with Fujitsu's DA is limited to selected local machines, which requires the entire heuristic, including the beam search, to be run on this machine. This in turn affects the runtime of our approach and complicates the comparison with the other approaches in terms of execution time. Therefore, we mark our time in italics in Table 1 and also report results for LKH3, GORT-GLS and BCP on our local machine used for the 3-phase heuristic to provide some more comparability regarding the computational time. The evaluation of our approach as well as the baseline models is done on datasets containing 100 randomly generated instances following the same data distribution as the training data.

Table 1: Mean cost, gap and time (sec.) per instance for different problem sizes

Model	$n = 20$			$n = 50$			$n = 100$		
	cost	gap (%)	time	cost	gap (%)	time	cost	gap (%)	time
3-Phase Heuristic	5.95	0.00	*10.18*	10.22	0.30	*80.08*	15.10	1.45	*153.53*
Gurobi	5.95	0.00	0.60	10.27	1.63	51.61	17.92	18.47	300.00
BCP	5.95	0.00	1.07	10.10	0.00	10.68	14.89	0.00	119.87
GORT-GLS	5.95	0.00	10.00	10.13	0.33	100.00	15.12	1.56	300.00
LKH3	5.95	0.00	6.20	10.13	0.30	11.74	15.08	1.28	19.69
GORT-GLS local	5.95	0.00	20.00	10.16	0.58	100.00	15.68	3.99	300.00
LKH3 local	5.95	0.00	10.41	10.13	0.30	23.75	15.08	1.28	40.96
BCP local	5.95	0.00	3.31	10.10	0.00	19.30	14.89	0.00	227.17

Our 3-phase heuristic demonstrates optimal performance for small instance sizes. For instances with 50 nodes, the heuristic yields results comparable to GORT-GLS and LKH executed on a high-performance computer, while outperforming GORT-GLS on the same local machine in terms of solution quality given the same computation time. For medium-sized instances, Gurobi already

cannot solve most of the instances to optimality within the given time limit. The BCP method outperforms all heuristics for medium instances. However, it shows a larger relative increase in running time for large instances, whereas LKH and our heuristic exhibit much slower scaling in computation time. For $n = 100$ nodes, our heuristic surpasses GORT-GLS in both running time and quality on both local and high-performance setups, albeit producing slightly inferior results compared to the LKH heuristic. Gurobi is generally unable to solve large instances within the time limit and has a significantly worse optimality gap compared to the heuristics. While BCP continues to produce the best results, the computation time of our heuristic scales noticeably slower with respect to the increase of the input size, which shows the potential of our approach to offer good scalability for larger instances.

5 Discussion

With our proposed 3-phase heuristic, we provide a proof of concept of how to use specialized quantum-inspired computing hardware such as Fujitsu's Digital Annealer in combination with deep learning to solve combinatorial optimization problems with constraints that are difficult to solve directly with the current state of quantum(-inspired) computing hardware. Computational results show that this approach is promising, as it utilizes the strengths of quantum(-inspired) computing to potentially provide better scalability than state-of-the-art methods such as GORT and BCP while yielding near-optimal solutions for larger instances.

Future research could focus on improving our approach in various ways. First, optimizing the computation time could involve implementing the heuristic more efficiently in C++ instead of using Python. However, optimizing the calculation time on the DA itself remains challenging, as no convenient termination criterion can be set for the DA. Secondly, the structure of our approach allows each phase to be approached using different methods. For instance, since the aim of this approach was to test to what extent the DA can be involved in solving the CVRPTW, no alternatives for finding the clusters in step 1 were tested. Future research could experiment to see how other cluster algorithms perform in this context as well as employing different hardware and software for solving QUBOs.

Our work demonstrates the potential of leveraging quantum-inspired computing in conjunction with deep learning for complex combinatorial optimization tasks. As quantum and quantum-inspired technologies continue to evolve, the integration of diverse computational techniques and hardware has the potential to provide even more efficient and versatile solutions for complex combinatorial optimization problems.

Disclosure of Interests. Shaglel and Kliesch are co-financed by the ERDF of the European Union and by Fonds of the Hamburg Ministry of Science, Research, Equalities and Districts (BWFGB). Kliesch is also funded by the Fujitsu Germany GmbH as part of the endowed professorship "Quantum Inspired and Quantum Optimization."

References

1. Bent, R., Van Hentenryck, P.: Spatial, temporal, and hybrid decompositions for large-scale vehicle routing with time windows. In: Cohen, D. (ed.) Principles and Practice of Constraint Programming – CP 2010. Lecture Notes In Computer Science, vol. 6308, pp. 99–113 (2010). https://doi.org/10.1007/978-3-642-15396-9_11
2. Bertsimas, D., Dunn, J.: Optimal classification trees. Machine Learning **106**(7), 1039–1082 (2017). https://doi.org/10.1007/s10994-017-5633-9
3. Bresson, X., Laurent, T.: Residual gated graph convnets. Preprint **arXiv:1711.07553** (2017). https://doi.org/10.48550/arXiv.1711.07553
4. Desaulniers, G., Madsen, O.B., Ropke, S.: The vehicle routing problem with time windows. In: Toth, P., Vigo, D. (eds.) Vehicle Routing: Problems, Methods, and Applications, vol. 2, pp. 119–159 (2014). https://doi.org/10.1137/1.9781611973594.ch5
5. Dondo, R., Cerdá, J.: A cluster-based optimization approach for the multi-depot heterogeneous fleet vehicle routing problem with time windows. European Journal of Operational Research **176**(3), 1478–1507 (2007). https://doi.org/https://doi.org/10.1016/j.ejor.2004.07.077
6. Dornemann, J.: Solving the capacitated vehicle routing problem with time windows via graph convolutional network assisted tree search and quantum-inspired computing. Frontiers in Applied Mathematics and Statistics **volume 9** (2023). https://doi.org/10.3389/fams.2023.1155356
7. Errami, N., Queiroga, E., Sadykov, R., Uchoa, E.: VrpSolverEasy: A python library for the exact solution of a rich vehicle routing problem. INFORMS Journal on Computing (2023). https://doi.org/10.1287/ijoc.2023.0103
8. Feld, S., Roch, C., Gabor, T., Seidel, C., Neukart, F., Galter, I., Mauerer, W., Linnhoff-Popien, C.: A hybrid solution method for the capacitated vehicle routing problem using a quantum annealer. Frontiers in ICT **volume 6** (2019). https://doi.org/10.3389/fict.2019.00013
9. Gillett, B.E., Miller, L.R.: A heuristic algorithm for the vehicle-dispatch problem. Operations Research **22**(2), 340–349 (1974). https://doi.org/10.1287/opre.22.2.340
10. Gurobi Optimization, L.: Gurobi optimizer reference manual (2022), https://www.gurobi.com
11. Helsgaun, K.: An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems: Technical report. Roskilde Universitet (2017), http://www.akira.ruc.dk/~keld/research/LKH-3/LKH-3_REPORT.pdf
12. Irie, H., Wongpaisarnsin, G., Terabe, M., Miki, A., Taguchi, S.: Quantum annealing of vehicle routing problem with time, state and capacity. In: Feld, Sebastianand Linnhoff-Popien, C. (ed.) Quantum Technology and Optimization Problems. QTOP 2019, Lecture Notes in Computer Science, vol. 11413, pp. 145–156 (2019). https://doi.org/10.48550/arXiv.1903.06322
13. Joshi, C.K., Laurent, T., Bresson, X.: An efficient graph convolutional network technique for the travelling salesman problem. Preprint **arXiv:1906.01227** (2019). https://doi.org/10.48550/arXiv.1906.01227
14. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. Preprint **arXiv:1412.6980v9** (2014). https://doi.org/10.48550/arXiv.1412.6980

15. Kowalsky, M., Albash, T., Hen, I., Lidar, D.A.: 3-regular three-XORSAT planted solutions benchmark of classical and quantum heuristic optimizers. Quantum Sci. Technol. **7**(2), 025008 (Apr 2022). https://doi.org/10.1088/2058-9565/ac4d1b
16. Laporte, G., Semet, F.: Classical heuristics for the capacitated vrp. In: Toth, P., Vigo, D. (eds.) The Vehicle Routing Problem, pp. 109–128 (2002). https://doi.org/10.1137/1.9780898718515.ch5
17. Lucas, A.: Ising formulations of many np problems. Frontiers in Physics **2** (2014). https://doi.org/10.3389/fphy.2014.00005
18. Matsubara, S., Takatsu, M., Miyazawa, T., Shibasaki, T., Watanabe, Y., Takemoto, K., Tamura, H.: Digital annealer for high-speed solving of combinatorial optimization problems and its applications. In: Yang, H., Cheng, K.T.T. (eds.) Proceedings of the 25th Asia and South Pacific Design Automation Conference. ASPDAC 2020. p. 667–672 (2020). https://doi.org/10.1109/ASP-DAC47756.2020.9045100
19. Ouyang, Y.: Design of vehicle routing zones for large-scale distribution systems. Transportation Research Part B: Methodological **41**(10), 1079–1093 (2007). https://doi.org/https://doi.org/10.1016/j.trb.2007.04.010
20. Perron, L., Furnon, V.: Or-tools routing library (2022), https://developers.google.com/optimization/
21. Poullet, J.: Leveraging machine learning to solve the vehicle routing problem with time windows. Master Thesis, Massachusetts Institute of Technology, Sloan School of Management, Operations Research Center (2020), https://hdl.handle.net/1721.1/127285
22. Qi, M., Lin, W.H., Li, N., Miao, L.: A spatiotemporal partitioning approach for large-scale vehicle routing problems with time windows. Transportation Research Part E: Logistics and Transportation Review **48**(1), 248–257 (2012). https://doi.org/10.1016/j.tre.2011.07.001
23. Rieffel, E.G., Venturelli, D., O'Gorman, B., Do, M.B., Prystay, E.M., Smelyanskiy, V.N.: A case study in programming a quantum annealer for hard operational planning problems. Quantum Information Processing **14**(1), 1–36 (2014). https://doi.org/10.1007/s11128-014-0892-x
24. Sadykov, R., Uchoa, E., Pessoa, A.: A bucket graph–based labeling algorithm with application to vehicle routing. Transportation Science **55**(1), 4–28 (2021). https://doi.org/10.1287/trsc.2020.0985
25. Solomon, M.M.: Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations Research **35**(2), 254–265 (1987). https://doi.org/10.1287/opre.35.2.254
26. Tran, T., Do, M., Rieffel, E., Frank, J., Wang, Z., O'Gorman, B., Venturelli, D., Beck, J.: A hybrid quantum-classical approach to solving scheduling problems. In: Baier, J.A., Botea, A. (eds.) Proceedings of the 9th International Symposium on Combinatorial Search. vol. 7, pp. 98–106 (2021). https://doi.org/10.1609/socs.v7i1.18390
27. Vinyals, O., Fortunato, M., Jaitly, N.: Pointer networks. In: Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 28, pp. 2692–2700 (2015), https://proceedings.neurips.cc/paper/2015/file/29921001f2f04bd3baee84a12e98098f-Paper.pdf