Proceedings of the ASME 2022 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference
IDETC/CIE 2022
August 14-17, 2022, St. Louis, USA

DETC2022/MESA-90123

A SCALABLE GRAPH LEARNING APPROACH TO CAPACITATED VEHICLE ROUTING PROBLEM USING CAPSULE NETWORKS AND ATTENTION MECHANISM

Steve Paul
Ph.D. Student,
Department of Mechanical
and Aerospace Engineering,
University Buffalo,
Buffalo, NewYork 14260
Email: stevepau@buffalo.edu

Souma Chowdhury
Associate Professor,
Department of Mechanical
and Aerospace Engineering,
University Buffalo,
Buffalo, NewYork 14260
Email: soumacho@buffalo.edu

ABSTRACT

This paper introduces a new graph neural network architecture for learning solutions of Capacitated Vehicle Routing Problems (CVRP) as policies over graphs. CVRP serves as an important benchmark for a wide range of combinatorial planning problems, which can be adapted to manufacturing, robotics and fleet planning applications. Here, the specific aim is to demonstrate the significant real-time executability and (beyond training) scalability advantages of the new graph learning approach over existing solution methods. While partly drawing motivation from recent graph learning methods that learn to solve CO problems such as multi-Traveling Salesman Problem (mTSP) and VRP, the proposed neural architecture presents a novel encoder-decoder architecture. Here the encoder is based on Capsule networks, which enables better representation of local and global information with permutation invariant node embeddings; and the decoder is based on the Multi-head attention (MHA) mechanism allowing sequential decisions. This architecture is trained using a policy gradient Reinforcement Learning process. The performance of our approach is favorably compared with state-of-the-art learning and non-learning methods for a benchmark suite of Capacitated-VRP (CVRP) problems. A further study on the CVRP with demand uncertainties is conducted to explore how this Capsule-Attention Mechanism architecture can be extended to handle real-world uncertainties by embedding them through the encoder.

1 Introduction

Combinatorial Optimization (CO) problems are ubiquitous in various real-world planning, scheduling and task allocation problems that are fundamental to the operation of complex cyber-physical systems. Often these CO problems can be naturally or with some modifications represented as a graph, thereby allowing the use of optimization, graph traversal, metaheuristics and machine learning tools that can operate over graphs, for solving the underlying CO problems. A majority of these problems tend to be NP-hard [1] and cannot be solved with polynomial time using traditional methods such as (Mixed) Integer (Non-)Linear Programming (ILP, MILP, MINLP, etc.) [2, 3], metaheuristics methods [4–7] and genetic algorithms [8–10]. CO problems such as Travelling Salesman Problem (TSP), Vehicle Routing Problem (VRP) can be used to model a wide variety of real world problems such as path planning for improved fuel economy in vehicles [10, 11], and manufacturing automation related applications such as pick and place [12], and sequence planning [13]. Even though some of these methods can generate local optimal solutions for small sized problems, the computational expense becomes intractable in applications where online near real-time (few milliseconds to few seconds) decisions are required. Some of the real-world systems applications of CO [14], such as network routing [15], transportation scheduling [16], power grid reconfiguration [17] and multi-agent task allocation [18], requires a near real-time performance; some of these applications also in-

volve a very large number of decision variables [19]. In most such application good feasible solutions that are near real-time computable is of essence, which becomes even more challenging in the face of uncertainties. A prototypical example would be planning of disaster relief [20] where the demand for relief (e.g., locations to be serviced or degree of service needed) is often uncertain; in such cases, a pre-computed plan based conventional CO techniques may not be able to provide the necessary performance and time-efficiency trade-offs [21].

In recent years, a rich body of work has emerged on using learning-based techniques to model solutions or intelligent heuristics for CO problems over graphs [22–27]. A notable fraction of these methods formulates the CO problem as a Markov Decision Process (MDP) and uses Reinforcement Learning (RL) to generate policies that can yield optimal solutions across a reasonable range of problem instances [22–25]. Such policies are often embodied by a trained graph neural network or GNN. Other methods have taken a supervised learning approach [28], with usage mostly for non-planning label-abundant applications. The main advantages of a learning approach over classical non-learning methods include [1]: **i)** the ability to generalize across problem scenarios without tedious hand-crafting of the heuristics; **ii)** much faster run-time execution (critical for real-time task allocation and planning applications); and **iii)** the ability (at least in theory) to automatically learn the problem features of interest which may not be readily evident even to human experts when applications involve complex cyber-physical systems. Benchmark applications have included Travelling Salesman Problem (TSP), Vehicle Routing Problem (VRP), and Max-Cut [22, 25]. Up to a certain scale and specificity of say planning and scheduling applications (as opposed to being able to generalize across wide range of problem classes), the complexity of such problems may not necessarily favor a learning based approach over non-learning based optimization, graph matching and local search heuristics methods [29, 30]. Another major limitation of most existing learning-based methods is their inability to readily apply a trained model on problem scenarios of greater complexity (e.g., multi-TSP instances with more travellers and/or locations to visit) than the scenarios or samples used for training; typically retraining is required to achieve this, which can quickly become computationally burden-some. Thus scalability with minimal to no re-training is a critical need for learning based solutions to be effective in modeling solutions to such CO problems.

In this paper we focus on a specific type of VRP, also known as the Capacitated VRP (CVRP) [31]. The CVRP is a combinatorial optimization problem where the objective is to compute the optimal route to deliver packages of the same size by a vehicle (with a maximum capacity on the number of packages that can be accommodated at a time) to a set of locations based on the given demand of the locations. The CVRP which is a generalized form of the VRP has is being used to model a wide variety of real-world applications such as for cyber-physical systems [32, 33], operations planning [34, 35]. A classification of the different methods for solving VRP can be found in [36]. The VRP has been traditionally tackled using exact MILP solutions such as given by Branch and bound [37–39], Branch and cut [40], Branch and price [41, 42], and Branch and cut and price [43] methods. Due to the NP-hard nature of the VRP, these methods become intractable for larger sized problems. On the other hand, heuristic based methods have included two phase routing [44], construction heuristics [45], Insertion heuristics [46, 47], and Iterative improvement heuristics [48]. A wide variety of meta-heuristic methods have also been modified and adapted to solve VRP, notable among which are Simulated Annealing [49], Genetic algorithms [50–52], Ant colony Optimization [53, 54], and Iterative local search [55]. Some of the learning based solutions to routing problems [22, 24, 25] demonstrate exceptional generalizability (performance on test data are comparable to a MILP based solution), with small computation time (for implementing the policies). A CVRP, which can be expressed as a graph [22, 24], as also discussed in section 2.1, presents rich structural information (beyond just Euclidean node coordinate properties), which can be exploited for computing better policies embodied by graph neural networks. While RL based methods such as [22, 24, 25] have to some extent demonstrated generalizability across unseen scenarios of similar complexity as those used in training, (we posit that) limited use of graph structural information in them has restrained their ability to scale without the need to retrain.

In this paper, we present a new *GNN-based encoder-decoder* neural architecture which acts as a policy network that can output the best action w.r.t. a state in a sequential manner. This neural architecture seeks to preserve local and global structural information, and is invariant to permutation of the node ordering in the graph representation of the problem. These characteristics are hypothesized to enable learning optimal policies for different classes of CO problems such that the learnt policies remain effective when applied to scenarios that are more complex (larger scale) than the problem scenarios used for training. Specifically we construct and explore a **Capsule Attention-based Model** trained through Reinforcement Learning, which we call **CapAM-RL**. It is composed of an encoder based on capsule networks, a decoder based on Multi-head attention (MHA) mechanism, and a context module that encapsulates and feeds the state input for a given type of problem. We use the CVRP to investigate the primary conceived benefit of our CapAM-RL architecture, namely how learning of global and local structural information improve the generalizability of a trained model over larger-sized problems (i.e., more complex than training samples), in comparison to state-of-the-art learning based methods [22]. To provide rigorous context and evidence of the roughly two-orders of magnitude faster run-time performance compared to non-learning based methods, we also compare the results with well-known local search and metaheuristics methods. Additional parametric analysis is presented to provide insights into the im-

pact of various parameters that control the node encoding, on the performance of the CapAM-RL architecture. We also demonstrate how our method can be extended to include a CVRP with demand uncertainties, and the corresponding advantage of explicitly embedding these uncertainties to yield more robust decision making.

Thus, the major contributions of this paper are as follows:

1. A new graph reinforcement-learning approach, CapAM-RL, which uses a novel Capsule Network based encoder and MHA based decoder is presented to learn policies for CVRP, and demonstrates the ability to generalize well, scale to larger size problem without the need to retrain, and is near-real-time executable even for large problems involving 1000's of nodes.
2. Comparative analysis of our approach in terms of generalizability, scalability and computation time, with other state-of-the-art learning and non-learning based approaches.
3. Extension of our CapAM-RL approach to be implemented on CVRP problems with demand uncertainty by embedding uncertainty parameters along with other node features under the graph formulation of the CVRP, and show how performance differs compared to the original (uncertainty agnostic) CapAM-RL.

The remainder of the paper is organized as follows: The next section describes the background and related works. Section 3 presents our proposed architecture and learning approach. Section 4 describes the CVRP experimental evaluations, baseline methods used for comparison, and discusses the results and further parametric analyses. The paper ends with concluding remarks.

2 Problem Formulation

2.1 Formulation of CVRP and its graph representation

The CVRP is a generalized version of the classical Vehicle Routing Problem (VRP). Each vehicle in the CVRP has a maximum capacity (C) constraint on the number of packages it can carry, and thus deliver in a single trip. Assuming there are N locations, each location i has a specific demand c_i on the number of packages, where $c_i \leq C$ and $i \in [1,N]$. Each task is designated an id between 1 to N. We also consider a depot with id as 0. The CVRP implementation in this work assumes a single vehicle with multiple trips (where each trip begins at the depot and ends in the depot to refill or for end of the mission) instead of multiple vehicle with single trip each. In this experiment, we do not consider split delivery where the demand of a location is fulfilled partially during a trip, and then completed in another trip. The decision can be perceived as a sequence of ordered list of location indices ($\phi_j,\ \ j \in [1,R]$), where j represents the j-th trip and R is the total number of trips made by the vehicle. Here, each ϕ_j can be expressed as a vector of location indices, e.g., $\phi_2 = [2\ 6\ 4\ 0]$ shows that in the 2nd trip, the vehicle visited the locations $i = 2$, $i = 6$ and $i = 4$ in that order before returning to the depot (given by index $i = 0$). The ballpark formulation for this CVRP can then be expressed as:

$$\min_{\phi_j,\ j\in[1,R]} f_{\text{cost}} = \sum_{j\in[1,R]} \mathscr{S}_j; \qquad (1)$$

$$\text{s.t. } C_{j,t+1} = \begin{cases} C_{j,t}, \text{ if } i \in \mathscr{V}, \text{ where } i = \phi_{j,t} \\ C, \text{ if } i = 0, \text{ where } i = \phi_{j,t} \\ \max(0, C_{j,t} - c_i), \text{ otherwise} \end{cases}$$

where $\mathscr{S}_j$ is the total distance travelled in trip j, C_t is the available capacity at the decision instance t, $\mathscr{V}$ is the set of locations already visited and demands fulfilled, and $\phi_{j,t}$ represents the t-th location visited in the j-th trip. The MILP formulation of the CVRP considered in the paper can be found in [31].

CVRP formulated as graphs: The task space of an CVRP can be represented as a complete un-directed graph $\mathscr{G} = (V, E, \mathbf{A})$, which contains a set of nodes/vertices (V), a set of edges (E) that connect the vertices to each other, and the weighted adjacency matrix $\mathbf{A}$ that gives the extent to which two nodes are connected. Each node is a task, and each edge connects a pair of nodes. For CVRP with N tasks, the number of vertices and the number of edges are N and $N(N-1)/2$, respectively. Node i is assigned a 3-dimensional feature vector denoting the node location and demand, i.e., $\delta_i = [x_i, y_i, c_i]$. Here we consider a weighted adjacency matrix without self-loop ($\alpha_{ii} = 0, \alpha_{i,j} \in \mathbf{A}$, where i and $j \in [1,N]$). The weights in the adjacency matrix α_{ij} take a real value between 0 and 1, in a manner such that if the features of two nodes i and j are relatively close to each other, then α_{ij} has a higher value. Since we are formulating CVRP as an un-directed graph, the adjacency matrix A is symmetric. In this paper, the elements of $\mathbf{A}$ ($\alpha_{i,j}$, $i \neq j$) are computed as the inverse of the normalized euclidean distance of the node properties. In addition, we also define a weight matrix (Ω) comprising the weights associated with each edge in the graph, where the weight of the edge connecting two task nodes ($\omega_{ij} \in \Omega$) represents the cost (e.g., distance traveled) incurred by the vehicle to visit node-j after node-i. Depending on the application, the weight matrix need not be symmetric.

2.2 MDP formulation for CVRP

The Markov Decision Process (MDP) of the CVRP problem can be defined to capture the myopic decision-making process for the vehicle, which can be expressed as a tuple $< \mathscr{S}, \mathscr{A}, \mathscr{P}_a, \mathscr{R} >$. The components of the MDP can then be defined as follows: **State Space ($\mathscr{S}$):** The vehicle uses a state

$s \in \mathscr{S}$, which contains the following information: 1) current location, 2) constraints the vehicle such as its current capacity. The state space also includes the environment, defined by the location and the demand of the location, which are expressed here as node features in the graph. **Action Space ($\mathscr{A}$):** The set of actions is represented as $\mathscr{A}$, where each action $a \in \mathscr{A}$ is defined as the index of the selected task, $\{1,\ldots,N\}$. In CVRP that includes a depot, designated as task 0. The location status, i.e., *active* or *completed* (i.e., location visited and demand is fulfilled), is used to properly decode the actions generated by the policy model. **Transition:** The transition is an event-based trigger. An event is defined as the condition that the vehicle reached its selected location or depot. **Reward ($\mathscr{R}$):** A delayed reward is considered here, estimated at the end of the simulation, i.e, when there are no more active locations to be visited. The actual reward function is typically application-dependent.

3 Architecture & Learning Framework

In this work we implement an RL algorithm on an **encoder-decoder** architecture to learn the optimal policies for CO problems over a graph formulation. The **encoder** represents each graph node as a continuous vector infusing both its own properties as well as its local and global structural properties within the graph representation of the node space. The decoder sequentially computes output probabilities for all the nodes using the information from the encoder and the current state (context), based on the Multi-Head Attention mechanism. The sequentially computed output probabilities for each node indicates the relative goodness of selecting that node as the next one to visit. Detailed description of the architecture, including the encoder, decoder and context modules, is presented in Section 3.1 to 3.3, based on the representative Capacitated VRP or CVRP problem described next.

3.1 Encoder

The main purpose of the encoder is to represent useful information related to a node as a continuous vector or tensor, which can then be used by the learning algorithm. For a graph node, the information includes the properties of the node itself (e.g., the coordinates and time deadline of the node), the local neighborhood information for the node, and also the global structural information. Combinatorial optimization problems such as CVRP, and MTSP are permutation invariant, which means the order by which each node is numbered should not affect the optimal solution. Hence the node encoding must also be permutation invariant. In this work, we are exploring how a Graph Capsule Convolutional Neural Network can be implemented for learning local and global structures with the node properties, with permutation invariant node embedding.

Graph Capsule Convolutional Neural Networks: Graph Capsule Convolutional Neural Networks (GCAPCN) is a class of Graph Neural Networks (GNN), introduced by [56] to address the drawbacks (e.g., permutation invariance) of Graph Convolutional Neural Networks (GCN), and to enable the encoding of global information based on a capsule idea presented in [57]. The main advantage of GCAPCN lies in capturing more local and global information, compared to conventional aggregation operations used in GNN such as aggregation or standard convolution operations. As described by [56], higher order statistical moments are used to compute a capsule vector, which captures local information better. Let $X \in \mathbb{R}^{N\times|\delta_i|}$, be the node feature matrix, where $|\delta_i|$ is the input dimension for each node i. The standard graph Laplacian is defined as $L = D - A \in \mathbb{R}^{N\times N}$, where D is the degree matrix of the graph. As described in [56], a capsule vector is computed using a Graph Capsule function based on different order of statistical moments, as shown in the equations later in this section.

Instead of directly feeding in the node properties [56], we first compute a feature vector F_{0i} for each node by linear transformation of the node properties δ_i, as $F_{0i} = \delta_i W_0 + b_0$ for all $i \in [1,N]$, where $W_0 \,\varepsilon\, \mathbb{R}^{h_0\times|\delta_i|}$, $b_0 \,\varepsilon\, \mathbb{R}^{h_0\times 1}$, and h_0 is the length of the feature vector. For a CVRP problem, $\delta_i = [x_i, y_i, c_i]$, where x_i, y_i, and c_i are the x coordinate, y coordinate, and the demand of location/node i. It is important to understand that this method for encoding is not limited to a specific CVRP problem. This approach can also be extended or modified for complex CO problems with nodes having other features.

Each feature vector $F_{0i}, i \in [1,N]$ is then passed through a series of Graph capsule layers. In each layer, the output from the previous layers is used to compute a matrix $f_p^{(l)}(X,L)$ using a graph convolutional filter of polynomial form as shown in Eq. 2.

$$f_p^{(l)}(X,L) = \sigma\Big(\sum_{k=0}^{K} L^k (F_{(l-1)}(X,L)^{\circ p}) W_{pk}^{(l)}\Big) \tag{2}$$

Here L is the graph Laplacian, p is the order of the statistical moment, K is the degree of the convolutional filter, $F_{(l-1)}(X,L)$ is the output from layer $l-1$, $F_{(l-1)}(X,L)^{\circ p}$ represents p times element-wise multiplication of $F_{(l-1)}(X,L)$. Here, $F_{(l-1)}(X,L) \in \mathbb{R}^{N\times h_{l-1}p}$, $W_{pk}^{(l)} \in \mathbb{R}^{h_{l-1}p\times h_l}$, and $f_p^{(l)}(X,L) \in \mathbb{R}^{N\times h_l}$. The output of layer l is obtained by concatenation of all $f_p^{(l)}(X,L)$, as shown in Eq. 3.

$$F_l(X,L) = [f_1^{(l)}(X,L), f_2^{(l)}(X,L), \ldots f_P^{(l)}(X,L)] \tag{3}$$

Here P is the highest order of statistical moment, and h_l is the node embedding length of layer l. We consider all the values of h_l (where $l \in [0, L_e]$) to be the same for this paper. Equations 2

and 3 were computed for L_e layers, where each layer uses the output from the previous layer ($F_{l-1}(X,L)$). Adding more layers helps in learning the global structure, however this can affect the performance by increasing the number of learnable parameters (compared to the size of the problem), leading to over-fitting. We also implemented batch normalization for regularization and skip connection to speed-up learning, between layers. The output from the final layer is then passed through a feed-forward layer so that the final feature vector has the right dimension to be fed into the decoder as shown in Fig. 1a. It is important to understand that this encoding approach can be easily extended to more complex problems. For example, for problems with locations having a time deadline, this will also be a part of the node properties (δ_i).

3.2 Decoder

The main purpose of the decoder is, to sequentially compute a probability distribution for the next node among all the available nodes, given the current state (or the context), and the node embeddings. This output probability of each node shows the importance of each node for that current state, or the importance of each node embedding for the context. The decoder implements an attention mechanism where the output from the context is used a query Q, and the node embeddings from the encoder are used as the key value pairs (K,V).

$$\text{Attention}(Q,K,V) = \text{softmax}(QK^T/\sqrt{d^k})V \tag{4}$$

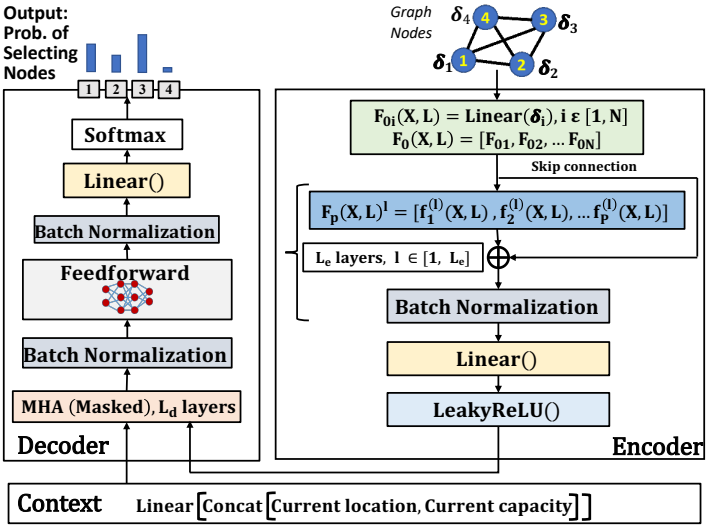
where d^k is the dimension of K or V. In this work we implement a multi-head attention (MHA) layer in order to determine the compatibility of Q with K and V. As shown by [58] the MHA layer can be defined as:

$$\text{Multihead}(Q,K,V) = \text{Linear}(\text{Concat}(\text{head}_1 \ldots \text{head}_h)) \tag{5}$$

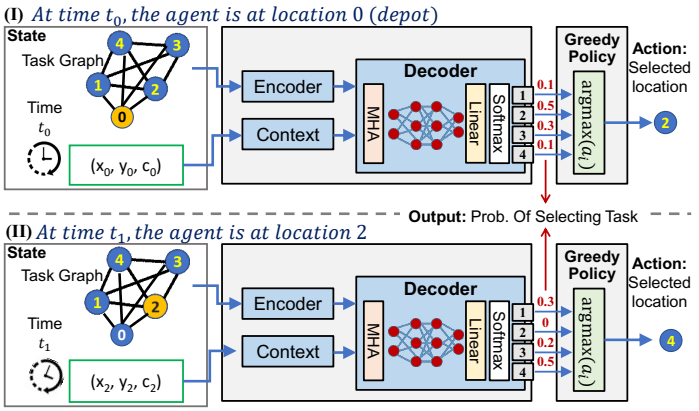
where $\text{head}_i = \text{Attention}(Q,K,V)$ and h is the number of heads. The feed-forward layer is to convert the output from the MHA layer to a dimension that can be taken in for the next process. The final softmax layer outputs the probability values for all the nodes. The next task to be done will be chosen based on a greedy approach, which means the node which has the highest probability will be chosen. The nodes which are already visited will be masked such that it will not be chosen in the future time steps of the simulation. During each decision making process the nodes whose demand cannot be met at that particular point of time, are also masked.

3.3 Context

The main function of the context is to encode the current state of the mission into a continuous vector, which can then be fed to the decoder to estimate the output probabilities. For a CVRP, the context consists of information regarding the current node (embedding), and the current vehicle capacity. These two pieces of information are concatenated, and a linear transformation was performed to obtain the query vector Q, which is then fed into the decoder. For a more complex problem the context includes more state variables that can affect the mission. For example, if the vehicle in the CVRP problem has a maximum distance range constraint, then it will also be added as context information, thus showing the versatility to be extended to more complex problems.



(a) CapAM-RL architecture: encoder, decoder and context



(b) Deployment of a CVRP policy based on CapAM-RL. I) at time t_0. II) at time t_1; here, the output for previously selected node, e.g., node 2 in (II), is set as 0.

FIGURE 1: Architecture of the Capsule Attention-based Model (CapAM) and how it is used for sequential decision-making

3.4 RL framework

For ease of implementation and fair comparison with another popular graph learning baseline [22] (that also uses REINFORCE), we implement the simple REINFORCE algorithm to train the CapAM-RL architecture in this paper. A pseudo code of this learning implementation is shown in Algorithm 1. Other relevant details for the training are discussed in section 4.2. However, the training process can readily be further advanced in the future through the adoption of more effective state-of-the-art policy gradient algorithms such as Proximal Policy Optimization [59] and the Actor-Critic method [60].

Algorithm 1: Learning Algorithm

Input: N_{E}: Number of epochs, N_b: Number of batch, B: Batch size, N_{tr}: Training data size, N_{vl}: Validation data size.

$\theta_{\text{CapAM-RL}}$ - CapAM-RL
$\theta^{BL}_{\text{CapAM-RL}}$ - Baseline CapAM-RL
for epoch = 1..N_{epoch} **do**
  $\mathscr{D}_{\mathrm{tr}}, \mathscr{D}_{\mathrm{vl}} \leftarrow \text{GenerateScenarios}(N_{\mathrm{tr}}, N_{\mathrm{vl}})$
  $N_b \leftarrow \lfloor N_{\mathrm{tr}}/B \rfloor$
  for step = 1..N_b **do**
    $\mathscr{D}_{\mathrm{tr,b}} \leftarrow \text{SampleRandom}(\mathscr{D}_{\mathrm{tr}}, M)$ {$\mathscr{D}_{\mathrm{tr,b}}$: Batch Training Dataset}
    $\mathbf{a}^{\mathrm{BL}}$, $f^{\mathrm{BL}}_{\text{cost}} \leftarrow \text{CalculateCost}(\theta^{BL}_{\text{CapAM-RL}}, \mathscr{D}_{\mathrm{tr,b}})$
    $\mathbf{a}$, $f_{\text{cost}} \leftarrow \text{CalculateCost}(\theta_{\text{CapAM-RL}}, \mathscr{D}_{\mathrm{tr,b}})$
    $\nabla\mathscr{L} \leftarrow \frac{1}{B}\sum_{i=1}^{B}(f_{\text{cost,i}} - f^{\mathrm{BL}}_{\text{cost,i}}) \log\text{softmax}(\mathbf{a}_{\mathrm{i}})$
    $\theta_{\text{CapAM-RL}} \leftarrow \textsc{Adam}(\nabla\mathscr{L}, \theta_{\text{CapAM-RL}})$
  end for
  $\mathbf{a}^{\mathrm{BL}}_{\mathrm{vl}}$, $f^{\mathrm{BL}}_{\text{cost,vl}} \leftarrow \text{CalculateCost}(\theta^{BL}_{\text{CapAM-RL}}, \mathscr{D}_{\mathrm{vl}})$
  $\mathbf{a}$, $f_{\text{cost}} \leftarrow \text{CalculateCost}(\theta_{\text{CapAM-RL}}, \mathscr{D}_{\mathrm{vl}})$
  if $(\sum_{i=1}^{N_{\mathrm{vl}}} f^{\mathrm{BL}}_{\text{cost,i}} > \sum_{i=1}^{N_{\mathrm{vl}}} f_{\text{cost,i}}) \wedge (\text{T-Test}(\mathbf{a}_{\mathrm{vl}}, \mathbf{a}^{\mathrm{BL}}_{\mathrm{vl}}) > \varepsilon)$ **then**
    $\theta^{BL}_{\text{CapAM-RL}} \leftarrow \theta_{\text{CapAM-RL}}$
  end if
end for
CalcuateCost Procedure:
for $i = 1..|\mathscr{D}|$ **do**
  $\mathbf{a}_{\mathrm{i}}, f_{\text{cost,i}} \leftarrow \text{Simulation}(\theta, \mathscr{D}_i)$
  $\mathbf{a} \leftarrow \mathbf{a} \cup \mathbf{a}_{\mathrm{i}}$
  $f_{\text{cost}} \leftarrow f_{\text{cost}} \cup f_{\text{cost,i}}$
end for
return $\mathbf{a}, f_{\text{cost}}$

4 Experimental Evaluations & Results on CVRP

4.1 Baseline methods

For a comprehensive comparative analysis, we compare our CapAM-RL method with existing state-of-the-art methods, including both learning based and state-of-the-art heuristic based (non-learning) methods. The heuristic based methods includes 1) Guided Local Search (**GLS**) and 2) Simulated Annealing (**SA**). Both these methods are implemented using the Google OR tools. We consider two learning based baseline methods. The first one is the Multi-head Attention based Mechanism with RL (**AM-RL**) reported by [22], which uses a similar encoder-decoder concept as CapAM-RL. The second learning baseline also follows the same encoder-decoder architecture with RL, but with a Graph Convolutional Networks [61] (**GCN-RL**) as the encoder.

To train and test AM-RL, we use the source code provided by [22]. To ensure fair and consistent comparison, we train and test all three learning-based methods (i.e., AM-RL, GCN-RL, and our CapAM-RL) using the same environment and simulation code (based on [22]). This is also essential to allow us to study how the three different types of encoders perform compared with each other. All three methods are trained with the same training algorithm (REINFORCE) and hyperparameters (further details can be found in Section 4.2).

4.2 Training and testing details

The *"Python"* 3.7 and the 64-bit distribution of *"Anaconda 2020.02"* are used to implement the CVRP approaches. The environment, the architecture, training algorithm, and the evaluation of the trained model, are all implemented in *Pytorch-1.5*. With the help of Pytorch, the training was deployed on two GPUs (NVIDIA tesla v100-pcie-16gb) with 16 GB RAM. The testing of all the trained models, and other baseline methods were implemented in a 2.6 GHz Intel core i7 MacOS 11.2.3 system for CVRP.

Training hyperparameters: We use REINFORCE algorithm with Rollout baselines for training all the models in this paper. Every model was trained for a total of 100 epochs. Each epoch consists of 500000 scenarios randomly generated from its corresponding distributions. The number of scenarios for the Rollout baseline is 10000. For each epoch, the model was updated using a batch size of 500 (due to memory constraints). The learning step size was set as 0.001 (from [22]). The optimizer used for updating the model is Adam optimizer.

Training Settings: We analyse the performance of CapAM-RL on generalizability, scalability and run-time, and then further conduct a parametric study to identify the best hyperparameters that provide the best performing (trained) model for each learning-based model. For training AM-RL, we use the recommended number of attention heads (n_h = 8) in the encoder and trained for different number of encoding layers (L_e = [1,2,3]). We use the model which performed the best in all test scenarios (L_e = 3) for comparing with CapAM-RL. Similarly, we train different models of GCN-RL (by varying the order of the convolutional filter K, and the number of layers), and use the best performing model (K = 3, L_e = 2) for comparison. The best model for each method was determined by the same approach as that described in Section 5. For our CapAM-RL method, we also conduct a parametric study and identify the best performing model (K = 3, P = 4, and L_e = 1), which will be used for comparative analysis. We train all three methods (AM-RL, GCN-RL, and CapAM-RL) on CVRP with 100 nodes, and then test the trained models on problems of different sizes, but from the same distribution as that

of the training data. The node embedding length (h) for all three methods is set at 128. Convergence plot for all the models used for comparison for CVRP, trained for 100 epochs is shown in Fig. 2 (for CapAM-RL), and Figures 6, 8 for AM-RL and GCN-RL resepectively, in appendix 5.

Dataset: The dataset used for training CVRP consists of scenarios with 100 locations and one depot. The x and y coordinates of the locations (including the depot) are randomly generated from a uniform distribution within the limits [0, 1]. The demand for all task locations will be a random integer from a uniform distribution between [1,9], with depot assigned a 0 demand. The vehicle capacity (C) for a scenario with 100 locations, is considered as 50. The dataset used for testing (to analyze both generalizability and scalibility) has the same limits as explained above. The assumed capacity of the vehicle for test scenarios of different number of locations are shown in Table 1. Each CVRP scenario is defined a single sample. The test cases consists of a total of 100 scenarios for all different number of locations (50, 100, 200, 500, 1000, and 2000 nodes).

TABLE 1: The capacity of the vehicle for different test scenarios

# of Tasks	Capacity (C)
20	20
50	40
100	50
200	100
500	250
1000	500
2000	1000

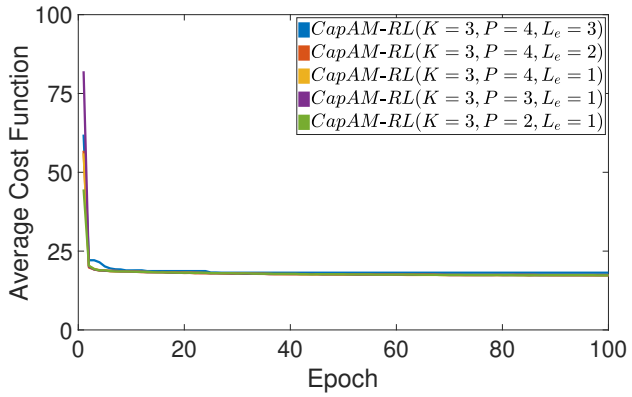
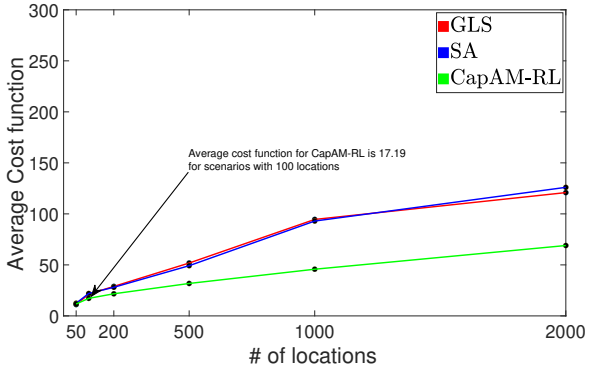


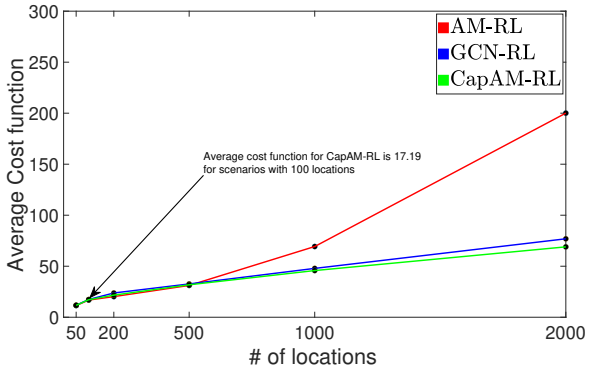
FIGURE 2: Convergence plot for training of different CapAM-RL models corresponding to each of the learning-based methods for CVRP. The number of locations for the training cases is 100

4.3 Generalizability and Scalability:

Generalizability analysis: In this paper, *generalizability* refers to the performance of the trained model on unseen test scenarios that involve the same number of locations as in the scenarios used for training; and where the test and training scenarios are drawn from the same probability distribution over locations. In this work, generalizability was analysed on test scenarios with the number of locations fixed at 50 and 2000, drawn from the same distribution over a 2D space. Figure 2 shows the convergence plot for CapAM-RL models with different parameters (in terms of P and L_e), where all the CapAM-RL models converges to a value of about 18, which is very close to the average cost function value of the 100 locations test cases for CapAM-RL, thus showing the generalisability of the method.



(a) With non-learning based baseline methods. Lower the better.



(b) With learning based baseline methods. Lower the better.

FIGURE 3: Performance comparison of CapAM-RL with different baseline methods. There are 100 test cases for each number of locations. Here CapAM-RL uses $K = 3$, $P = 4$, and $L_e = 1$ (best performing model), and $h = 128$ for all learning based methods.

Scalability analysis: In this analysis, we study if learning the global and local structural information helps to learn generalizable optimal policies over problems of increasing/decreasing scale compared to the original problem(s) over which the policies were trained. For this purpose, we use the models which were trained for 100 tasks, and implement them on unseen problems scaled up by a factor f_s. This scalability study is performed for f_s

= 2, f_s = 5, f_s = 10, and f_s = 20. We compared the scalability of our CapAM-RL method with heuristic-based and learning-based methods and the results are shown in Figs. 3a-3b.

Figure 3a shows that CapAM-RL (in terms of the cost function) outperforms significantly the non-learning based methods for larger-sized problems, although both GLS and SA has a slightly better performance for small-sized problems. This clearly shows the advantage of learning-based approaches compared with heuristic-based methods.

As it can be seen from Fig. 3b, all the learning-based methods have comparable performance on problems with < 500 nodes, with slightly better performance by AM-RL. However, for the scenarios with larger nodes (> 500), the performance of AM-RL drops drastically compared to CapAM-RL and GCN-RL. This observation shows the inability of the MHA-based encoder to generalize on larger sized problems and demonstrates how the capsule networks can generalize on larger sized problem by using more global and local information. A parametric analysis on how the local and global information can impact the performance of the trained model is provided in Section 4.4.

The sequential decision-making process using CapAM-RL is shown in Fig. 1b. Both for cost computation while training, and when using the trained model to determine the node sequence for test cases, we implement a sequential decision making process; the decisions are produced by the decoder. This sequential process is explained in Algorithm 1, and used over the complete simulation of an episode, where each CVRP case in the training and testing dataset is considered as an episode. Figure 10 shows the routes generated by CapAM-RL for three different scenarios with 50 locations.

TABLE 2: Comparing average run-time (**seconds**) to generate solutions for CVRP test cases.

	Non-learning		Learning		
# nodes	**GLS**	**SA**	**AM-RL**	**GCN-RL**	**CapAM-RL**
50	4	4	0.04	0.04	0.04
100	8	8	0.09	0.08	0.09
200	12	12	0.18	0.16	0.17
500	50	50	0.53	0.51	0.53
1000	100	100	1.51	1.41	1.49
2000	200	200	4.95	4.61	4.81

Runtime analysis: Table 2 shows the average time taken by each method to generate the entire sequence. Since both CapAM-RL and AM-RL executes the learned policies to generate the sequence, the runtime is less compared to GLS and SA. It is important to note that both GLS and SA might generate a better solution than the one reported here by increasing the runtime. However, due to time constraint and for fair comparison we set the runtime as in table 2.

4.4 Parametric study:

One of the main factors affecting the performance of our approach is the amount of local and global information being infused into the node embeddings. This depends on the number of layers (L_e) in the encoder, the highest order of statistical moment (P), and the node embedding length (h). We trained our model with different values of L_e, P, and h. As explained by [56], the impact of K and L_e is the same, hence we did not performed a study varying K. All the models for this study were trained using the same data distribution as explained in section 4.1. Since the main contribution of this paper on encoding the node information, we are limiting the parametric study for the encoder parameters, and not on the learning algorithm or the decoder.

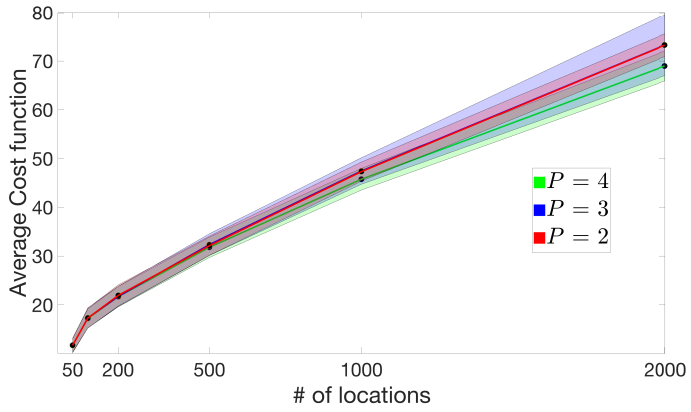
Varying number of layers L_e: For studying the impact of the number of layers on the performance we trained our mode by varying L_e from 1 to 3, keeping other parameters constant. Figure 4b shows the average cost function for models with varying L_e. It can be seen that L_e = 3 has better performance which is more evident for cases with larger number of nodes. It is also interesting to note that model with L_e = 1 performing better than L_e = 2.

Varying highest order of statistical moments P: For this study we trained our model with varying highest order of statistical moment P, while keeping other parameters constant. Figure 4a shows the performance comparison for the trained models for this study. Model with P = 4 has better performance than P = 3 and P = 2, especially for larger number of nodes

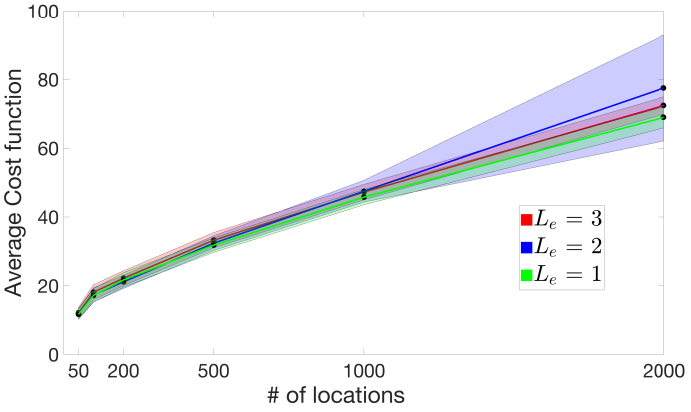
Influence of node embedding: The three learning based methods discussed in this paper (AM-RL, GCN-RL, and CapAM-RL) represents three different node embedding algorithms. Therefore this comparison (Fig. 3b) shows the quality of the node embedding methods for scaling to problems of larger size. The MHA encoder in AM-RL exhibits better performance for problems with sizes (up to 200 nodes) close to what it was trained for (100 nodes), but its performance decreases drastically for larger sized problems. The main difference between GCN and GCAPCN is that in GCN the convolutional operation results in a scalar (which results in loss of significant structural information as discussed by [56]), while for GCAPCN it is a vector of length P (discussed in Appendix 5). The performance for lower number of nodes is very close for all the three encoder, but for higher number of nodes, GCAPCN has a clear advantage, can be seen in Fig. 3b.

4.5 CVRP with demand uncertainty

In order to demonstrate the capabilities of our proposed method when there exists some form of uncertainty, we imple-



(a) Effect of P on the performance. $K = 3$, $L_e = 1$, $h = 128$ for all the models.



(b) Effect of L_e on the performance. $K = 3$, $P = 4$, $h = 128$ for all the models.

FIGURE 4: Parametric analysis of CapAM-RL. In both the figures, the shaded area corresponding to each color represent the standard deviation.
Lower the better.

ment our method on a CVRP with uncertainty in the demand at every location. We modify the CVRP environment in such a way that the demand at every locations changes following a Gaussian distribution. The demand at each location is different at different point of time. During each decision making process, the agent receives the updated demand information and here we assume that this demand at the location chosen during a decision making process does not change till the vehicle reaches the location and satisfies the demands. This is a safe assumption following demand modelling study [62, 63].

4.5.1 Demand uncertainty modeling and Graph formulation

Here we assume that the demand follows a Gaussian distribution similar to the demand distribution considered in [62], and the demand distribution at each location is independent of the other locations. Each location i is characterized by it's $x-y$ coordinates (x_i, y_i), the mean demand of the location c_i, and it's corresponding standard deviation σ_i. These locations can be considered as the nodes of a fully connected graph similar to the graph formulation for a normal CVRP as described in section 2.1. Therefore, the node properties can be represented as $\delta_i = [x_i, y_i, c_i, \sigma_i]$.

Architectural changes: The only architectural change that has to be made is to include the mean demand and the corresponding standard deviation to the node properties as described in the above section (section 4.5.1). The remaining procedure to compute the node embeddings remain the same (as described in section 3.1).

4.5.2 Training and testing details

The training and testing datasets are generated following the description in section 4.2. The only additional parameter is the standard deviation σ_i for the demand of all the locations $i \in [1, N]$. We consider the standard deviation of the demands to take random number between 0 and 2 with a uniform distribution.

4.5.3 Performance analysis

For this study, we compare two CapAM-RL models where one of the model is trained with including the demand uncertainty (denoted by CapAM-unc) as a node property (as described in section 4.5.1), and another model which does not include demand uncertainty (denoted by CapAM) as a node property. For this study we considered the K, L_e, and P values for both the models to be the same as that of the best performing CapAM-RL model in the generalizability and scalability study in section 4.3. Figure shows the performance comparison of CapAM-unc vs CapAM. While the performance of the two models in almost the same for lower number of nodes (50, 100, and 200), for scenarios with larger number of nodes, the CapAM-unc demonstrates a better performance in terms of the cost function, with 18.4% and 4.6% improvement compared to CapAM. This demonstrates that including the demand uncertainty along with the other node properties has a clear advantage while computing policies under uncertainties. Even though it is not possible to comment on the optimality of the routes for the different scenarios for this study, however by this study we are able to demonstrate how our method can be extended to problems with uncertainties and also showing the improvement in the performance by incorporating the uncertainties while learning.

5 Conclusion

In this paper we introduce a new GNN-based encoder-decoder architecture for learning scalable policies for a class of CO problems, namely Capacitated Vehicle Routing Problems or CVRP (which is representative of various scheduling and planning problems in cyber physical systems). We demonstrate how

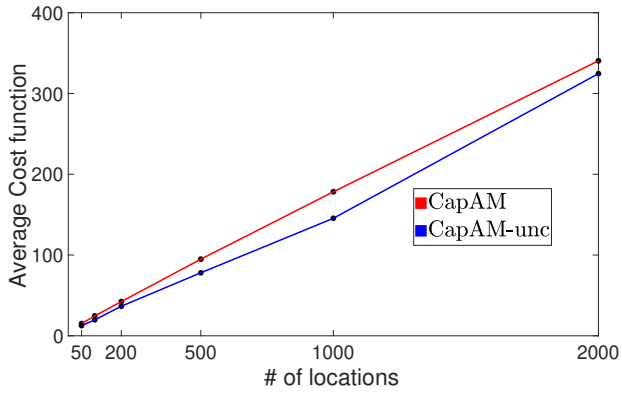


FIGURE 5: Comparison of CapAM-RL models on CVRP with demand uncertainty. CapAM-unc represents the model which is trained including demand uncertainties in the node embedding, while CapAM does not include the demand uncertainties.

GNN and RL can be combined to learn policies for a CVRP of a certain size (number of locations), such that the learned policy can predict actions which results in performance compared to other state-of-the-art baselines, at the same time demonstrate better performance than the baselines for larger sized problems, without the need to re-train the model. We empirically demonstrate how a novel node encoding, one that preserves the node properties and its global and local structural information, facilitates this scalability. More fundamental understanding of the impact of embedding such structural information (e.g., through graph geometry analysis) remains to be explored in future work. We show that our computation time for generating policies are much faster than some of the industry standard heuristic methods, thus establishing the potential to be applied for problems that require near real-time decision-making. It should be noted that our method is not limited to a CVRP, and can be extended to more complex multi-agent CO problems such as Multi-Agent Task Allocation and Production Planning. The sequential decision making along with learning the local and global structural information, makes CapAM-RL an excellent candidate for decision making under uncertainty. Furthermore, we show that even though more structural information is being encoded by increasing the value of P (highest order of statistical moment) and L_e (number of layers), this also leads to an increase in the number of trainable weights, which can in-turn deteriorate the performance. The study on the CVRP with demand uncertainties shows how the CapAM-RL can be extended to incorporate demand uncertainties, and how the inclusion of the uncertainty parameters yield better performance under testing.

Future directions: This work is implemented for a predefined number of nodes or task locations, but in principle could be extended for cases where nodes are added dynamically (i.e., where all demand locations are not known apriori). Parametric analysis of the learning algorithm, as well as the implementation of more recent state-of-the-art RL algorithms (e.g., PPO), can be considered as other directions of future work with CapAM-RL. Currently the CapAM-RL yielded policy model makes a greedy choice based on the predicted node-to-visit selection probabilities. To allow a better balance between exploration/exploitation, an epsilon greedy approach could be explored in the future, which along with application to real-world planning problems is expected to further elucidate the potential of this new graph learning paradigm.

ACKNOWLEDGMENT

This work was supported by the Office of Naval Research (ONR) award N00014-21-1-2530 and National Science Foundation (NSF) award 2048020. Any opinions, findings, conclusions, or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the ONR or the NSF.

REFERENCES

[1] Peng, Y., Choi, B., and Xu, J., 2021. "Graph learning for combinatorial optimization: A survey of state-of-the-art". *Data Science and Engineering*, pp. 1–23.

[2] Miller, C. E., Tucker, A. W., and Zemlin, R. A., 1960. "Integer programming formulation of traveling salesman problems". *Journal of the ACM (JACM),* ***7***(4), pp. 326–329.

[3] Kamra, N., and Ayanian, N., 2015. "A mixed integer programming model for timed deliveries in multirobot systems". In 2015 IEEE International Conference on Automation Science and Engineering (CASE), IEEE, pp. 612–617.

[4] Rizzoli, A. E., Montemanni, R., Lucibello, E., and Gambardella, L. M., 2007. "Ant colony optimization for real-world vehicle routing problems". *Swarm Intelligence,* ***1***(2), pp. 135–151.

[5] Wang, X., Choi, T.-M., Liu, H., and Yue, X., 2016. "Novel ant colony optimization methods for simplifying solution construction in vehicle routing problems". *IEEE Transactions on Intelligent Transportation Systems,* ***17***(11), pp. 3132–3141.

[6] , 2012. A New Heuristic for Traveling Salesman Problem Based on LCO, Vol. ASME/ISCIE 2012 International Symposium on Flexible Automation of *International Symposium on Flexible Automation*.

[7] Scanlon, R., Wang, Q., and Wang, J., 2016. "Ant Colony Optimisation model for vehicle routing problem with simultaneous pickup and delivery". In Proceedings of the ASME Design Engineering Technical Conference.

[8] Zhang, T., Gruver, W., and Smith, M. H., 1999. "Team

scheduling by genetic search". In Intelligent Processing and Manufacturing of Materials, 1999. IPMM'99. Proceedings of the Second International Conference on, Vol. 2, IEEE, pp. 839–844.

[9] Mühlenbein, H., 1991. "Parallel genetic algorithms, population genetics and combinatorial optimization". In Parallelism, Learning, Evolution, J. D. Becker, I. Eisele, and F. W. Mündemann, eds., Springer Berlin Heidelberg, pp. 398–406.

[10] Zhu, G. G., and Miao, C., 2019. "Real-Time Co-optimization of Vehicle Route and Speed Using Generic Algorithm for Improved Fuel Economy". *Mechanical Engineering,* ***141***(03), 03, pp. S08–S15.

[11] Chirala, V. S., Venkatachalam, S., Smereka, J. M., and Kassoumeh, S., 2021. "A Multi-Objective Optimization Approach for Multi-Vehicle Path Planning Problems Considering Human–Robot Interactions". *Journal of Autonomous Vehicles and Systems*.

[12] Xie, L., Li, X., and Wei, X., 2022. "A Two-layer Heterogeneous Ant Colony System with Applications in Part-Picking Planning". *Journal of Computing and Information Science in Engineering*, 05, pp. 1–18.

[13] Gundupalli, S. P., Shukla, R., Gupta, R., Hait, S., and Thakur, A., 2021. "Optimal Sequence Planning for Robotic Sorting of Recyclables from Source-Segregated Municipal Solid Waste". *Journal of Computing and Information Science in Engineering*.

[14] Grötschel, M., Krumke, S. O., Rambau, J., Winter, T., and Zimmermann, U. T., 2001. "Combinatorial online optimization in real time". *Online optimization of large scale systems*, pp. 679–704.

[15] Mammeri, Z., 2019. "Reinforcement learning based routing in networks: Review and classification of approaches". *IEEE Access,* ***7***, pp. 55916–55950.

[16] Khadilkar, H., 2018. "A scalable reinforcement learning algorithm for scheduling railway lines". *IEEE Transactions on Intelligent Transportation Systems,* ***20***(2), pp. 727–736.

[17] Gao, Y., Wang, W., Shi, J., and Yu, N., 2020. "Batch-constrained reinforcement learning for dynamic distribution network reconfiguration". *IEEE Transactions on Smart Grid,* ***11***(6), pp. 5357–5369.

[18] Lin, K., Zhao, R., Xu, Z., and Zhou, J., 2018. "Efficient large-scale fleet management via multi-agent deep reinforcement learning". In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 1774–1783.

[19] Shetye, K. S., Overbye, T. J., Li, H., Thekkemathiote, J., and Scribner, H., 2021. "Considerations for interconnection of large power grid networks". In 2021 IEEE Power and Energy Conference at Illinois (PECI), pp. 1–8.

[20] Ghassemi, P., and Chowdhury, S., 2020. "Multi-Robot Task Allocation in Disaster Response: Addressing Dynamic Tasks with Deadlines and Robots with Capacity Constraints". *Robotics and Autonomous Systems*(under review), dec.

[21] Cappart, Q., Chételat, D., Khalil, E. B., Lodi, A., Morris, C., and Veličković, P., 2021. "Combinatorial optimization and reasoning with graph neural networks".

[22] Kool, W., Van Hoof, H., and Welling, M., 2019. "Attention, learn to solve routing problems!". In 7th International Conference on Learning Representations, ICLR 2019.

[23] Barrett, T. D., Clements, W. R., Foerster, J. N., and Lvovsky, A. I., 2019. "Exploratory combinatorial optimization with reinforcement learning". *arXiv preprint arXiv:1909.04063*.

[24] Khalil, E., Dai, H., Zhang, Y., Dilkina, B., and Song, L., 2017. "Learning combinatorial optimization algorithms over graphs". In Advances in Neural Information Processing Systems, pp. 6348–6358.

[25] Kaempfer, Y., and Wolf, L., 2018. "Learning the multiple traveling salesmen problem with permutation invariant pooling networks". *ArXiv,* ***abs/1803.09621***.

[26] Li, Z., Chen, Q., and Koltun, V., 2018. "Combinatorial optimization with graph convolutional networks and guided tree search". In Advances in Neural Information Processing Systems, pp. 539–548.

[27] Nowak, A., Villar, S., Bandeira, A. S., and Bruna, J., 2017. "A note on learning algorithms for quadratic assignment with graph neural networks". *stat,* ***1050***, p. 22.

[28] Mittal, A., Dhawan, A., Manchanda, S., Medya, S., Ranu, S., and Singh, A., 2019. "Learning heuristics over large graphs via deep reinforcement learning". *arXiv preprint arXiv:1903.03332*.

[29] Jose, K., and Pratihar, D. K., 2016. "Task allocation and collision-free path planning of centralized multi-robots system for industrial plant inspection using heuristic methods". *Robotics and Autonomous Systems,* ***80***, pp. 34–42.

[30] Vansteenwegen, P., Souffriau, W., Vanden Berghe, G., and Van Oudheusden, D., 2009. "Iterated local search for the team orienteering problem with time windows". *Computers & Operations Research,* ***36***(12), pp. 3281–3290. New developments on hub location.

[31] Ralphs, T. K., Kopman, L., Pulleyblank, W. R., and Trotter, L. E., 2003. "On the capacitated vehicle routing problem". *Mathematical programming,* ***94***(2), pp. 343–359.

[32] Mishra, A., and Ray, A. K., 2020. "Iot cloud-based cyber-physical system for efficient solid waste management in smart cities: a novel cost function based route optimisation technique for waste collection vehicles using dustbin sensors and real-time road traffic informatics". *IET Cyber-Physical Systems: Theory & Applications,* ***5***(4), pp. 330–341.

[33] Bhatia, V., Jaglan, V., Kumawat, S., and Kaswan, K. S., 2022. "Real-life applications of soft computing in cyber-

physical system: A compressive review". *Soft Computing: Theories and Applications*, pp. 501–514.

[34] Bakhtiari, A., Navid, H., Mehri, J., Berruto, R., and Bochtis, D., 2013. "Operations planning for agricultural harvesters using ant colony optimization". *Spanish Journal of Agricultural Research,* ***11***(3), pp. 652–660.

[35] Ali, O., and Van Oudheusden, D., 2009. "Infield logistics planning for harvest operations". *Engineering optimization,* ***41***(2), pp. 24–34.

[36] Toro O, E. M., Escobar Z, A. H., and Granada E, M., 2016. "Literature review on the vehicle routing problem in the green transportation context". *Luna Azul*(42), pp. 362–387.

[37] Laporte, G., and Nobert, Y., 1987. "Exact algorithms for the vehicle routing problem**the authors are grateful to the canadian natural sciences and engineering research council (grants a4747 and a5486) and to the quebec government (fcac grant 80eq04228) for their financial support.". In *Surveys in Combinatorial Optimization*, S. Martello, G. Laporte, M. Minoux, and C. Ribeiro, eds., Vol. 132 of *North-Holland Mathematics Studies*. North-Holland, pp. 147–184.

[38] Fischetti, M., Toth, P., and Vigo, D., 1994. "A branch-and-bound algorithm for the capacitated vehicle routing problem on directed graphs". *Operations Research,* ***42***(5), pp. 846–859.

[39] Baldacci, R., Hadjiconstantinou, E., and Mingozzi, A., 2004. "An exact algorithm for the capacitated vehicle routing problem based on a two-commodity network flow formulation". *Operations Research,* ***52***(5), pp. 723–738.

[40] Cordeau, J.-F., 2006. "A branch-and-cut algorithm for the dial-a-ride problem". *Operations Research,* ***54***(3), pp. 573–586.

[41] Pessoa, A., de Aragão, M. P., and Uchoa, E., 2008. *Robust Branch-Cut-and-Price Algorithms for Vehicle Routing Problems*. Springer US, Boston, MA, pp. 297–325.

[42] Martinelli, R., Poggi, M., and Subramanian, A., 2013. "Improved bounds for large scale capacitated arc routing problem". *Computers & Operations Research,* ***40***(8), pp. 2145–2160.

[43] Baldacci, R., Toth, P., and Vigo, D., 2010. "Exact algorithms for routing problems under vehicle capacity constraints". *Annals of Operations Research,* ***175***(1), pp. 213–245.

[44] Beasley, J. E., 1983. "Route first—cluster second methods for vehicle routing". *Omega,* ***11***(4), pp. 403–408.

[45] Clarke, G., and Wright, J. W., 1964. "Scheduling of vehicles from a central depot to a number of delivery points". *Operations Research,* ***12***(4), pp. 568–581.

[46] Mole, R., and Jameson, S., 1976. "A sequential route-building algorithm employing a generalised savings criterion". *Journal of the Operational Research Society,* ***27***(2), pp. 503–511.

[47] Laporte, G., Gendreau, M., Potvin, J.-Y., and Semet, F., 2000. "Classical and modern heuristics for the vehicle routing problem". *International transactions in operational research,* ***7***(4-5), pp. 285–300.

[48] Gendreau, M., Hertz, A., and Laporte, G., 1992. "New insertion and postoptimization procedures for the traveling salesman problem". *Operations Research,* ***40***(6), pp. 1086–1094.

[49] Alfa, A. S., Heragu, S. S., and Chen, M., 1991. "A 3-opt based simulated annealing algorithm for vehicle routing problems". *Computers & Industrial Engineering,* ***21***(1-4), pp. 635–639.

[50] Lacomme, P., Prins, C., and Ramdane-Chérif, W., 2001. "A genetic algorithm for the capacitated arc routing problem and its extensions". In Workshops on applications of evolutionary computation, Springer, pp. 473–483.

[51] Jorgensen, R. M., Larsen, J., and Bergvinsdottir, K. B., 2007. "Solving the dial-a-ride problem using genetic algorithms". *Journal of the Operational Research Society,* ***58***(10), pp. 1321–1331.

[52] Minocha, B., Tripathi, S., and Mohan, C., 2011. "Solving vehicle routing and scheduling problems using hybrid genetic algorithm". In 2011 3rd international conference on electronics computer technology, Vol. 2, Ieee, pp. 189–193.

[53] Donati, A. V., Montemanni, R., Casagrande, N., Rizzoli, A. E., and Gambardella, L. M., 2008. "Time dependent vehicle routing problem with a multi ant colony system". *European journal of operational research,* ***185***(3), pp. 1174–1191.

[54] Atefi, R., Iori, M., Salari, M., and Vezzali, D., 2022. A practical vehicle routing problem for monitoring water distribution networks.

[55] François, V., Arda, Y., and Crama, Y., 2019. "Adaptive large neighborhood search for multitrip vehicle routing with time windows". *Transportation Science,* ***53***(6), pp. 1706–1730.

[56] Verma, S., and Zhang, Z. L., 2018. Graph capsule convolutional neural networks.

[57] Hinton, G. E., Krizhevsky, A., and Wang, S. D., 2011. "Transforming auto-encoders". In Artificial Neural Networks and Machine Learning – ICANN 2011, T. Honkela, W. Duch, M. Girolami, and S. Kaski, eds., Springer Berlin Heidelberg, pp. 44–51.

[58] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I., 2017. "Attention is all you need". *CoRR,* ***abs/1706.03762***.

[59] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O., 2017. Proximal policy optimization algorithms.

[60] Konda, V., and Tsitsiklis, J., 2003. "Onactor-critic algorithms". *SIAM J. Control. Optim.,* ***42***, pp. 1143–1166.

[61] Kipf, T. N., and Welling, M., 2016. "Semi-supervised

classification with graph convolutional networks". *arXiv preprint arXiv:1609.02907*.

[62] Jabali, O., Rei, W., Gendreau, M., and Laporte, G., 2014. "Partial-route inequalities for the multi-vehicle routing problem with stochastic demands". *Discrete Applied Mathematics,* ***177***, pp. 121–136.

[63] Mendoza, J. E., Castanier, B., Guéret, C., Medaglia, A. L., and Velasco, N., 2011. "Constructive heuristics for the multicompartment vehicle routing problem with stochastic demands". *Transportation science,* ***45***(3), pp. 346–363.

Appendix A: Further information on node encoding

Node Embeddings in GCAPCN vs GCN

The graph convolutional operation in a GCN is the aggregation of node information corresponding to each feature. One of the major drawbacks of this operation as pointed by [57], is the potential loss of structural information of the graph nodes.

To elaborate on the above point, let G be a graph with N nodes, $X \in \mathbb{R}^{N\times|\delta_i|}$, and L be the feature matrix and the graph Laplacian respectively, and $|\delta_i|$ is the input dimension of the node properties. For a GCN, the most general form of the output function of a layer l can be expressed as:

$$f^{(l)}(X,L) = \sigma(\sum_{k=0}^{K} L^k f^{(l-1)}(X,L)W_k^l) \tag{6}$$

where k is the degree of the convolutional filter (of polynomial form), $[W_1^l....W_K^l]$ are learning weight matrices with $W_k^l \in \mathbb{R}^{h_{l-1}\times h}$, $f^{(l)}(X,L) \in \mathbb{R}^{N\times h_l}$, h_l is the feature vector length or embedding length in layer l, and σ is the activation function. Therefore for a node i (assuming h_l features), each feature is expressed as a scalar.

The Graph Capsule Convolutional Neural Networks (GCAPCN) address this drawback by encapsulating more structural information of the nodes, where higher order statistical moments of the features are used for computing the feature vector for each layer as shown earlier in equation 2, where $p \in [1,P]$ is the order of the statistical moment. Therefore each feature is expressed as a vector of length P, instead of a scalar.

Convergence plots for AM-RL and GCN-RL models

Further supporting results for CVRP **Finding the best performing models for CapAM-RL, GCN-RL, AM-RL**: We train the three learning based methods with different parameters. Tables 3, 4, and 5 give the average computing time for one training epoch of CapAM-RL, GCN-RL and AM-RL, respectively. Tables 6, 7, and 8 show the mean and standard deviation of the testing results (CVRP cost function) for different models of the three methods. For each method we choose the model which has the best performance in a majority of the test problems. For example, from Table 6, it can be seen that CapAM-RL ($K = 3$, $P = 4$, L_e=1) performs the best in terms of cost function (smaller the better) for cases with 100, 500, 1000, and 2000 locations. For all the different sized problems, we use 100 test cases. Similarly for GCN-RL, the best performing model has $K = 3$, and $L_e = 2$ (7), while for AM-RL, the best performing model has $n_h = 8$, and L_e = 3 Table 8).

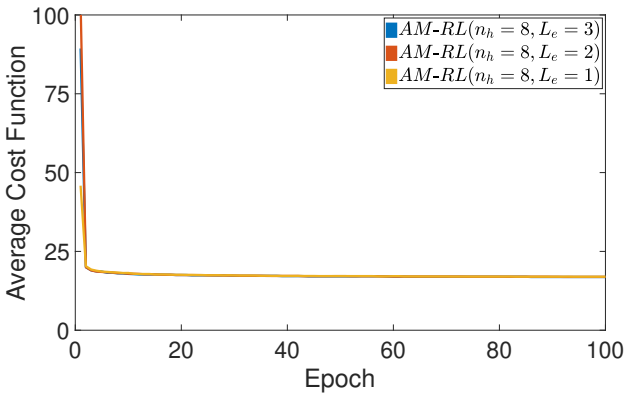


FIGURE 6: AM-RL

FIGURE 7: Convergence plot for training of different AM-RL models corresponding to each of the learning-based methods for CVRP

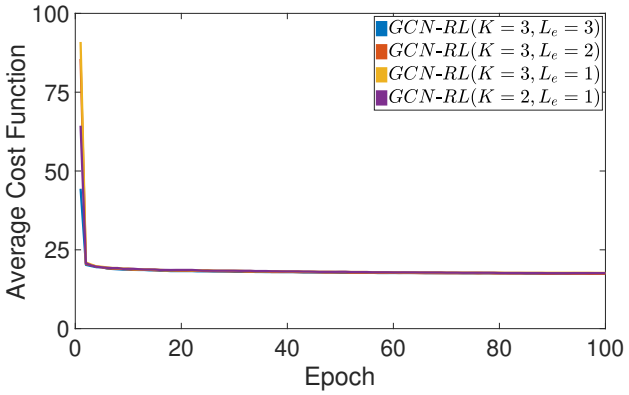


FIGURE 8: GCN-RL

FIGURE 9: Convergence plot for training of different GCN-RL models corresponding to each of the learning-based methods for CVRP

TABLE 3: Average time for finishing a single training epoch for different CapAM-RL models for CVRP

Method	CapAM-RL ($K=3$, $P=4$, $L_e=3$)	CapAM-RL ($K=3$, $P=4$, $L_e=2$)	CapAM-RL ($K=3$, $P=4$, $L_e=1$)	CapAM-RL ($K=3$, $P=3$, $L_e=1$)	CapAM-RL ($K=3$, $P=2$, $L_e=1$)
Average epoch time (mins)	17	16	14	16	13

TABLE 4: Average time for finishing a single training epoch for different GCN-RL models for CVRP

Method	GCN-RL ($K=3$,$L_e=3$)	GCN-RL ($K=3$, $L_e=2$)	GCN-RL ($K=3$, $L_e=1$)	GCN-RL ($K=2$, $L_e=1$)
Average epoch time (mins)	17	14	14	13

TABLE 5: Average time for finishing a single training epoch for different AM-RL models for CVRP

Method	**AM-RL** ($n_h = 8$,$L_e = 3$)	**AM-RL** ($n_h = 8$, $L_e = 2$)	**AM-RL** ($n_h = 8$, $L_e = 1$)
Average epoch time (mins)	18	18	17

TABLE 6: Average cost function (with standard deviation) of the different CapAM-RL models when implemented on the test data (of different number of locations). The values in each column shows the mean (standard deviation in brackets) cost function of the 100 test cases.

# locations	**CapAM-RL** ($K = 3$, $P = 4$, $L_e = 3$) **mean (std)**	**CapAM-RL** ($K = 3$, $P = 4$, $L_e = 2$) **mean (std)**	**CapAM-RL** ($K = 3$, $P = 4$, $L_e = 1$) **mean (std)**	**CapAM-RL** ($K = 3$, $P = 3$, $L_e = 1$) **mean (std)**	**CapAM-RL** ($K = 3$, $P = 2$, $L_e = 1$) **mean (std)**
50	12.12 (1.58)	11.849 (1.44)	11.67 (1.48)	**11.67(1.26)**	11.712 (1.44)
100	18.095 (2.18)	17.325 (2.03)	**17.20(1.96)**	17.313 (2.04)	17.272 (2.04)
200	22.176 (2.14)	**21.118 (1.91)**	21.676 (2.08)	21.714 (2.07)	21.913 (2.15)
500	33.224 (2.22)	32.369 (2.13)	**31.798 (2.08)**	32.345 (2.22)	32.121 (1.97)
1000	47.219 (2.23)	47.479 (3.17)	**45.755 (2.21)**	47.443 (2.71)	47.315 (1.95)
2000	72.429 (2.51)	77.536 (15.42)	**68.987 (3.07)**	73.308 (6.31)	73.25 (2.34)

TABLE 7: Average cost function (with standard deviation) of the different GCN-RL models when implemented on the test data (of different number of locations). The values in each column shows the mean (standard deviation in brackets) cost function of the 100 test cases.

# locations	**GCN-RL** ($K = 3$, $L_e = 3$) **mean (std)**	**GCN-RL** ($K = 3$, $L_e = 2$) **mean (std)**	**GCN-RL** ($K = 3$, $L_e = 1$) **mean (std)**	**GCN-RL** ($K = 2$, $L_e = 1$) **mean (std)**
50	11.85 (1.45)	**11.786 (1.42)**	11.952 (1.58)	11.848 (1.50)
100	17.425 (1.98)	**17.387 (1.98)**	17.6 (2.08)	17.528 (2.00)
200	23.695 (3.84)	23.778 (3.20)	24.506 (3.45)	**23.589 (2.45)**
500	33.201 (2.78)	**32.641 (2.17)**	33.995 (2.19)	34.118 (2.11)
1000	51.992 (10.53)	**47.878 (2.78)**	52.32 (5.80)	53.075 (8.36)
2000	97.447 (65.81)	**76.888 (17.38)**	101.73 (44.23)	94.191 (40.14)

TABLE 8: Average cost function (with standard deviation) of the different AM-RL models when implemented on the test data (of different number of locations). The values in each column shows the mean (standard deviation in brackets) cost function of the 100 test cases.

# locations	**AM-RL** ($n_h = 8$, $L_e = 3$) **mean (std)**	**AM-RL** ($n_h = 8$, $L_e = 2$) **mean (std)**	**AM-RL** ($n_h = 8$, $L_e = 1$) **mean (std)**
50	**11.438 (1.34)**	11.609 (1.39)	11.61 (1.38)
100	**16.828 (1.90)**	16.91 (1.97)	17.006 (1.92)
200	**20.229 (1.67)**	20.411 (1.86)	20.731 (1.74)
500	**31.16 (2.37)**	34.801 (3.68)	55.587 (24.20)
1000	**69.314 (18.92)**	81.213 (16.97)	249.35 (177.16)
2000	**200.07 (80.12)**	257.91 (70.45)	592.49 (386.06)

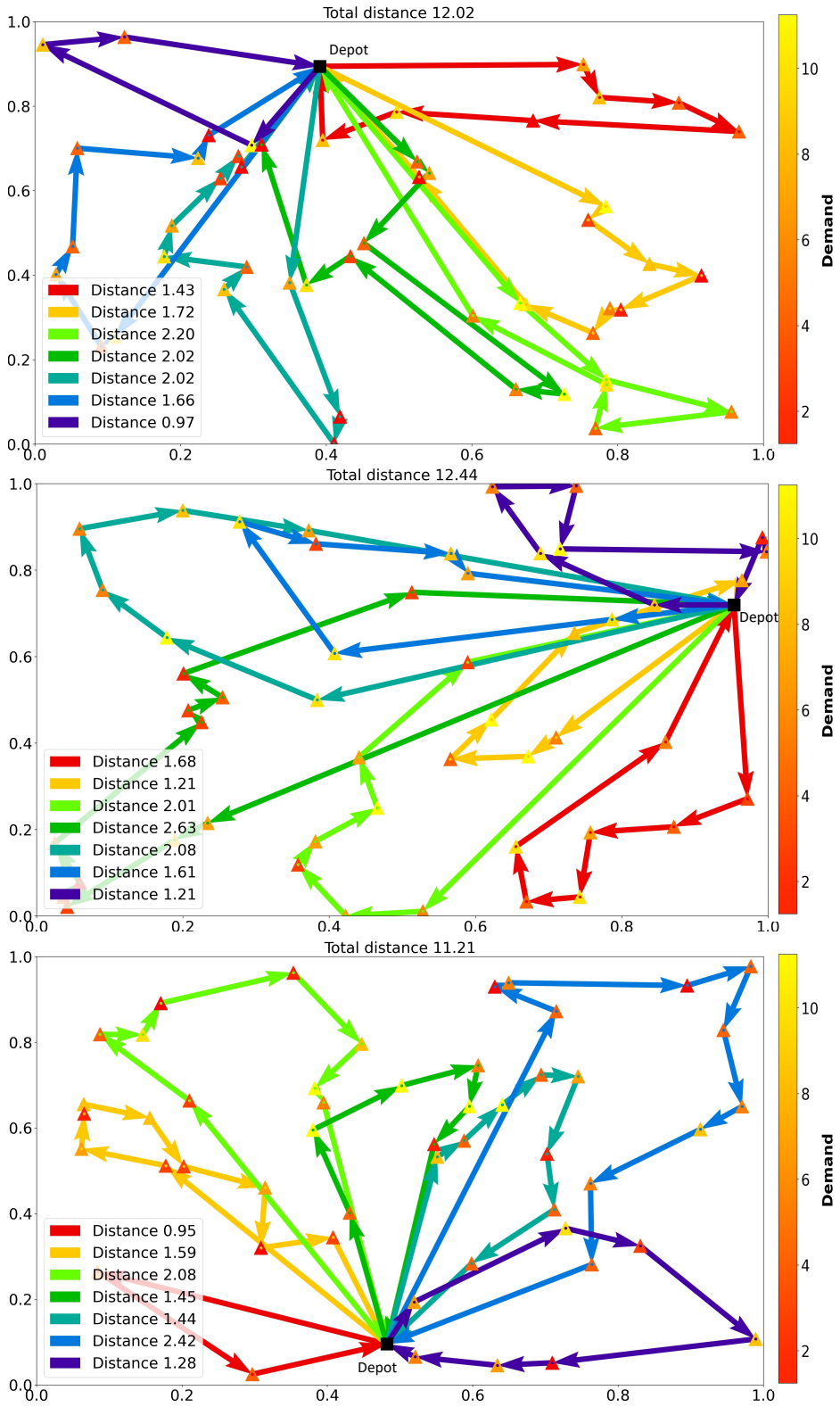


FIGURE 10: Routes generated by CapAM-RL for 3 different scenarios with 50 locations. Triangles represent task locations colored proportional to the local demand. The black square represents the depot location.