Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 212 (2022) 418–428

Procedia Computer Science

www.elsevier.com/locate/procedia

11th International Young Scientist Conference on Computational Science

Multi-Agent Reinforcement Learning For Multi Vehicles One-commodity Vehicle Routing Problem

Yamen Habib, Andrey Filchenkov*

ITMO University, Kronverksky pr. 49a, St. Petersburg 197101 Russia

Abstract

Vehicle Routing Problem with Dynamic and Stochastic information (DS-VPR) is mathematical problem reflecting a huge set of real-world logistic problem. In this paper, we presented a novel method to handle DS-VRP. Our approach depends on multi-agent reinforcement learning where we place our decision makers in nodes instead of vehicles and we use geometric Laplacian eigenmaps embedding to represent graph nodes information. We also presented a new method of training RL agents in a graph by separating it into two phases. Also, we built a simulation to develop and test different methods for the DS-VRP problem. The results showed that our approach shows the same performance being trained from the scratch as Google OR tools and MARDAM, while being more adaptive than these methods and having less parameters than MARDAM.

Keywords: vehicle routing problem; stochastic dynamic vehicle routing problem; multi-agent systems; deep reinforcement learning

1. Introduction

Modern society is significantly reliant on complex multi-modal logistic networks as part of a globalization trend. This is especially important for urban areas that are attracting an increasing number of residents and commercial activities. With the development of e-commerce, same-day delivery, externalized food delivery, cloud kitchens, and car- and ride-sharing, customers' consumption habits are changing, and new services are emerging. Such densely populated areas with significant economic players necessitate well-organized logistic distribution networks that operate on tighter schedules than ever before. Developing complex logistic networks while keeping in mind the social, economic, and environmental goals that shape our future is a challenging task for decision-makers and all stakeholders in the sector. Delivery vehicles, planes, boats, trucks, trains, and even bicycles are used to transport goods from one side of

* Corresponding author. Tel.: +7-950-021-0328;
E-mail address: afilchenkov@itmo.ru

1877-0509
10.1016/j.procs.2022.11.026

the globe to the other. Their synchronization and cost optimization involves a great deal of proper preparation, as well as reliable and robust recourse optimization.

New technologies in the fields of smart cities and autonomous cars give up a wide range of possibilities for enhancing the capabilities of such logistic systems. They provide real-time data on unpredictable element factors that can affect logistic operations, such as parking space availability. They also allow agents to make decisions in a decentralized manner, allowing them to react and adapt to local disturbances while still achieving a shared goal.

One of the major issues in Operations Research (OR) is assigning and ordering deliveries to ultimate customers at one end of the logistic chain, which has substantial social, ecological, and economic consequences. The **Vehicle Routing Problem (VRP)** is the generic abstract problem class that models multi-vehicle logistics. It has been extensively studied since the late 50s, and numerous extensions have been proposed to take into account a variety of operational restrictions, such as Capacitated Vehicles and Time Windows (CVRPTW). Classical techniques mostly centered on a static and deterministic formulation in which consumers and trip costs are known a priori, but in recent decades, variations of the issue incorporating some Dynamic and Stochastic information (DS-VRPs) have received growing attention. We focus on DS-VPRs as being more precise in reflecting real-world circumstances.

The OR community has been creating and refining techniques to solve the problem of determining routes that serve all clients for a fleet of vehicles while minimizing costs and obeying some limitations for the past sixty years. Exact techniques have achieved a level of performance where they can find routes for situations with hundreds of clients and show the optimality of the solutions. Meta-heuristics have been effectively employed to generate almost optimal outcomes for larger problems or those with more complex sets of constraints. However, current approaches for DS-VRPs largely rely on the same hand-crafted expert heuristics, which are used for static and deterministic problems, that do not fully exploit all of the available information, particularly past experiences from which the system could automatically learn specialized heuristics to generate near-optimal routes much faster online.

Reinforcement Learning (RL) offers potential methods to optimize policies that take actions conditioned on the state of the system in a sequential decision-making process when decisions must be modified dynamically to a stochastic environment. Exact RL approaches do not scale well with the dimensions of the task. That is why we require customized approximate representations that are constructed so that the subset of solutions we can investigate is consistent with the structure of the problems. As a result, this subset is more likely to include near-optimal solutions. Hopefully, the generalization capacity of Deep Neural Networks (DNNs) and their ability to learn complex non-linear functions have opened up new possibilities in the field of Deep RL, particularly in its application to combinatorial issues. The key challenges are creating meaningful vector representations of graph-structured data and efficiently adjusting for the problem restrictions. Despite this, all previous research in this area reframes the VRP as a single-vehicle problem.

This paper focuses on a multi-agent perspective of DSVRP. It also broadens the range of problems that Deep RL-based techniques may solve to include rich DSVRPs. But **to what extent this multi-agent reinforcement learning approach can help solve the issues encountered in DS-VRPs**? This research question inspired and guided our investigation. The contribution of this paper include:

— a novel method to handle DS-VRP based on multi-agent reinforcement learning with decision makers places in nodes represented with geometric Laplacian eigenmaps;

— a new method of training RL agents in a graph by separating it into two phases;

— a simulation to develop and test different methods for the DS-VRP problem.

2. Related work

The Vehicle Routing Problem (VRP) is a classic mathematical abstraction of a variety of real-world delivery problems. To the best of our knowledge, Dantzig et al. [11] were the first to introduce it in 1959, when they considered Capacitated vehicles VRPs (CVRPs), which have been extensively examined in the OR literature since then, and have been extended to account for a variety of operational restrictions, such as customers with Time Windows (VRPTW) or services such as Pickup and Delivery (PDP). In addition to these constraints, V. Pillac et al. [19] noted that the evolution (static or dynamic) and quality (deterministic or stochastic) of the information available are usually what distinguishes classical abstractions from real-world applications.

Static and deterministic VRPs can usually be formalized as (Mixed) Integer Linear Program. During past decades, many contributed to solving these problems either by exact methods such as Branch-and-Bound citetoth2002branch, or heuristics such as Genetic Algorithms [1], Ant Colony Optimization [4] or Large Neighborhood Search [14]. Dynamic VRPs (D-VRPs) require a solution model to adapt the routes of the vehicles online during their interactions with the environment (world). The most famous method to tackle this problem is either by re-optimizing an already defined solution or having a policy that is capable to provide decision rules for any possible state of environment [17].

Deep reinforcement learning (DRL) has recently produced a new class of heuristics [6]. The main idea is to learn to solve combinatorial problems from massive collections of data using the representation power of deep neural networks (DNNs). This approach faces some major challenges: effectively representing unordered groups of customers while taking into account the hard constraints of the problem; keeping track of each vehicle's distinct internal state [16]. These DRL techniques have been demonstrated to generate solutions that are competitive with those provided by handcrafted expert heuristics once they have been trained. Furthermore, they offer a distinct benefit over other heuristics in terms of running time during execution in exchange for the initial offline training time investment.

VPR has many variations. Although these variations have the same general goal of serving customers paying the lowest possible cost, they differ from each other with the set of constraints presented in each [12].

In **Capacitated Vehicle Routing Problem** [11], we consider a fleet of m vehicles, that have to serve n different customers. Customers must be assigned to vehicles, and each vehicle must be routed via its allocated customers. They all start at the same location (depot) and must return there at the end of their routes. In Capacitated VRP (CVRP), additionally, every customer i has a demand q_i that corresponds to the number of commodities (items) to provide. Every vehicle has a maximum capacity that allows it to service just a portion of the consumers. In **Pickup and Delivery Problem (PDP)** [15] and **Dial-A-Ride Problem (DARP)** [10], additionally, each customer i is connected with a pair of nodes: one for pickup labeled with a $+q_i$ demand, and one for delivery labeled with a $-q_i$ demand. **Dynamic and Stochastic CVRP (DS-CVRP)**, is the most general of these problems. Additionally, only a proportion of customers are known a priori before the vehicles leave their depots. While the vehicles are in movement, the remaining customers appear dynamically. The degree of dynamism $r_{dyn} \in [0, 1]$ of the problem is defined as the ratio between the number of unknown customers and the total number of customers.

Most approaches loosen the service restriction by allowing certain customers' requests to be ignored. The quality of service of the solution is defined as the ratio of the number of served customers to the total number of customers. Different approaches to balance this level of service against the cost of the solution exist. Some will predict and integrate some waiting points in their routes [20], relying on predefined strategies to serve newly arriving consumers in close proximity, which can be contrasted to sophisticated recourse actions. Other approaches will re-optimize routes for each new customer encountered, often keeping a pool of virtual anticipatory routes to keep things interesting and speed up the searching for a new solution [7]. Finally, some approaches will give rules that accept online observations of the state of the environment and build the routes progressively and dynamically by committing only to one delivery at a time [22], which is closer to our approach and an MDP formulation. It is worth noting that an Integer Linear Program is not well suited for describing this variation of the problem any longer. One may consider a new ILP every time a new customer appears, but this would require the introduction of extra state variables to account for previously committed deliveries and partial routes taken by the vehicles.

Vinyals et al. [25] proposed to use an attention mechanism to output a permutation of the input sequence. They trained their model to solve three combinatorial problems on graphs: Convex Hull, Delaunay Triangulation, and more importantly for us TSP. However, they could not outperform the heuristics they used to label the dataset. In [14] the architecture was extended in [5] by inserting some attention glimpses between the decoder and the final additive attention layer outputting the distribution on the input sequence. More importantly, they used a loss function borrowed from reinforcement learning to train their model directly from the performances of solutions sampled from the model. This self-improving approach enabled them to outperform the basic heuristics used as ground truth in [25].

In paper [13], Transformer Encoder [24] was integrated into an architecture designed to sample high-quality initial solutions for TSPs, which are then simply refined by a 2-opt heuristic. Another attempt has done in [15], where end-to-end heuristic, which uses Transformer Encoder and the Multi-Head Attention layer, was proposed to efficiently sample solutions for a variety of problems ranging from VRPs to Stochastic TSPs. Additionally to this architecture contribution, they introduce a Monte-Carlo baseline to reduce the variance of the gradient estimator during training. Their Attention Model (AM) learn to represent the graph of customers through a Transformer encoder. From the

encodings of the customers and depot, they create a context vector which gets updated after each decision. This context vector drives an attention mechanism which computes a score for each node, used as the probability to select it as next target.

The MARDAM architecture [9] was built to address rich DS-VRPs is inspired by and can be seen as an extension of AM. It uses the same attention mechanisms, namely Transformer and MHA, to encode the global state of customers and vehicles and build individual decision rules for every agent controlling the vehicles.

3. Environment simulation

In order to verify the results of routing algorithms in DS-VRP systems, first, it is necessary to have a test virtual environment. It will allow obtaining preliminary estimates of the quality of the proposed approaches before using them in the real world. We built our simulation using Simpy which is a process-based discrete-event simulation framework based on standard Python.

Starting from a problem instance, we need first to represent the city. As our customers are dynamic and not predefined, we chose to present the city as a graph with a predefined number of nodes that represent a potential future place for an upcoming request. In a real-life implementation, we will map new customers' locations to the nearest node from our city graph. Nodes are connected using edges labeled with a defined cost. In VRPs, we care the most about minimizing the cost of the passed edges, thus it is important to define the cost in a way to fulfill our needs. The cost might represent an approximation of the amount of fuel needed by the vehicle to be transported from one node to another. Another possible cost is the time of transportation or the distance between the nodes. One might think that these different costs have similar outcomes to the solution, but in real life, there are a lot of factors that play different roles in these costs like gas stations, traffic, and others. Therefore, most companies use a more complex function built by urban experts that consider all previous factors. In our implementation, we assign a constant random value between 1 and 10 for each edge.

Customers are represented by their requests. Every predefined number of time units a new request will be created. There are two types of requests: one with a positive quantity that indicates that we need to pick up this amount from the customer. The other with negative quantity indicates that we need to deliver this amount to the customer. In this work, we are interested only in one-commodity case, which means we do not distinguish between different customers' commodities and what we pick up from a customer, can be delivered to any other one. An example of this kind of problem is the ATM machines, where a bank just cares about keeping the amount of money in each ATM machine within a specific range, thus filling it up when there is a shortage and take away in case of the existence of the extra amount of money.

After describing the model of a city, and we need to represent the vehicles that will help us serve our customers. As it can be noticed from the city definition, we do not have a depot or a unique node to start and end vehicles' journeys. This is because we consider any node to be a start-end point for any vehicle. We will see later that the flexible environment's state observation gives us the ability to do this. We are interested in capacitated vehicle routing problem, therefor our vehicles have a predefined capacity limit.

Once a vehicle gets to a node with a request, the simulation engine will check if the current amount and the capacity conditions allow serving the request. Once the request is served, the status of the vehicle will be updated and the request will be deleted. The important note is that we do not restrict a node one-visit restriction as in static VRPs. Therefore, a vehicle can visit a node more than once and a node might host a request more than once also (simultaneously or not).

Every vehicle has two important attributes, weight-till-now and max-weight. Every time a vehicle passes an edge, weight-till-now will be updated according to some function (in the current implementation, we add the cost of the edge to the wight-till-now attribute). Once weight-till-now exceed max-weight, we end the current route of the vehicle. For convenience, we do not consider the cost to return to the start node of the vehicle in the decision-making process, but we add it to the final cost. The reason for this is that the vehicle returns to its start node from the node with the last request served by itself and for this, we can use any shortest-path algorithm.

During the process of serving, each vehicle goes through multiple states as shown in Fig. 2. First, each vehicle is enqueued in a node based on the attached configuration file. If the starting node has a request, it will be served, or the vehicle will be in the ready state. As long as there are no requests, the vehicle stays in the wait state. Once a

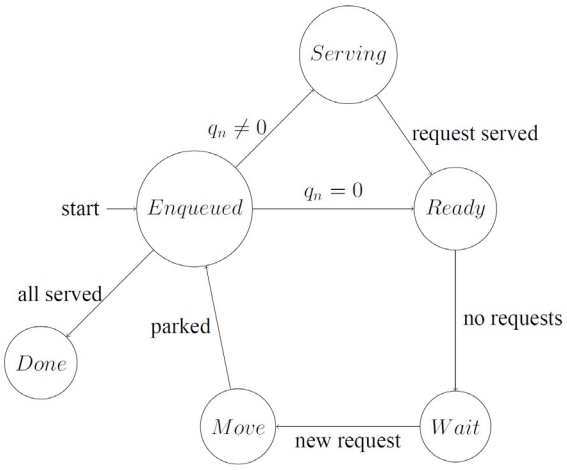

Fig. 1. States of the vehicles

new request is initiated, the vehicle starts moving based on the decision of the router in this node. When we serve all requests vehicle goes to the done state and returns to its start node. Vehicles pass one unit of weight (distance in our implementation) each one unit of time.

The final important thing to describe is the messages. Every node has a router and every router has an agent (e.g., the reinforcement learning agent). Every time the state of a vehicle changes, an event occurred and this event is sent to the corresponding router. The corresponding agent handles the event and sends a message. These messages are how our agents communicate with each other and play a crucial role in reinforcement learning as we use them to send the reward messages back to our agents.

4. Routing algorithm

As we mentioned earlier, DS-VRP is an NP-hard problem and any successful solution has to consider the major challenges that characterize the problem. The most important one is the ability to make a decision based on the online dynamic changes in the state of the environment. These changes might occur to the customers when the quantity a customer requests changes. This happens when a new customer appears and they have been assigned to the same node. Another change might happen to the topology of the graph, like the weight (cost) between two nodes. A simple example of this case is when we take the time of transportation as our cost and traffic between two nodes, then our agents need to know these changes and consider them when taking the routing decisions. However, in this work, we are only interested in the changes in the customers' states. States of the vehicles change with regularity, therefore our agents need to consider them also.

DS-VRP is a multi-dimensional problem that requires making several decisions for different aspects of the problem. The first one is **where to place our agents**. Here we have two main options. The first option is to place the agents in the vehicles. The second option is to place the agents in the nodes. In this work, we investigate how this option might succeed in handling DS-VRPs.

The second choice is **how to represent the data needed to make routing decisions**. Customers' locations might be represented as their geographic coordinates (latitude and longitude), or we can use some kind of graph embedding. This choice affects intensively the architecture of the neural network used by agents. We see later how the form of the output also affects the results of the agents. Next, we will explore these different aspects and we will see discuss how we made our choices.

4.1. Graph representation

Transforming a graph into a representation that can be directly put into a typical machine learning pipeline is one technique to apply existing machine learning methods to graphs. Graph representation learning, also known as graph embedding, aims to create a vector representation of a graph by assigning a feature vector to each node, which can then be later used by any machine learning algorithm. Graph embedding seeks to build a low-dimensional representation of a graph. This low-dimensional representation is then used for various downstream tasks.

Several VRP approaches tried to handle this problem using the neural networks model itself. In other words, they use the latitude and the longitude as part of the input of the neural network model [9]. This method complicates the training process. Another drawback to this method is that it depends on the exact values of the latitude and the longitude and not the relative location of the nodes to each other. Thus, the knowledge a model gets in one topology cannot be useful in another one even if it is similar to it.

A different approach is to use an auxiliary algorithm to represent as much information as possible from a graph. One popular method is Laplacian Eigenmaps [3], which constructs a graph embedding based on the spectral properties of the Laplacian matrix of the graph. The idea behind it, and many other embedding techniques, is that the embedding of a graph must respect node similarity: similar nodes must have embeddings that are close to one another. This technique overcomes the previous one in its ability to be generalized to similar typologies. More importantly, it separates the embedding of the graph from the neural network which leads to building a more task-oriented model and hopefully smaller ones with fast inference time.

Geometric Laplacian Eigenmap Embedding (GLEE) [23] uses the Laplacian matrix to find an embedding with geometric properties instead of spectral ones, by leveraging the so-called simplex geometry of the graph. In this work we consider GLEE as graph embedding for the following reasons:

- GLEE, while closely related to the Laplacian Eigenmaps (LE) method, outperforms LE in the tasks of link prediction and graph reconstruction. Moreover, a common critique of LE is that it only considers first-order adjacency in the graph, while GLEE takes into account higher order.
- GLEE outperforms existing graph embedding methods (which minimize the distance between similar nodes) which suffer when the graph average clustering coefficient is low.
- Many works have been done to address the problem of generalization when GLEE embedding is applied with multi-agent reinforcement learning. One recent work is FastGe [2] method, which is based on the wisdom of the crowd principle. The wisdom of crowds means that making decisions based on multiple sources of data or knowledge is more efficient than just using a single source.

4.2. Environment

One of the most important parts when handling DS-VRP is tracking the evaluation of information during the solution-building phase. As we are considering the reinforcement learning approach to handle the stochastic information, we need an efficient way to represent the current state of the environment. We have three main focuses that need to be presented in the state space:

- Embedding of the **node**, where the vehicle is parked: we need it to decide what is our next target and for this, we use GLEE embedding of the current node.
- Current customers' **requests**: for this, we build a matrix containing in its rows the embedding of all the nodes of our graph. We append to each row an element representing the quantity of the request of the customer. If the quantity is 0, it means no requests in this node.
- Current state of **vehicles**: we also build a matrix containing in its rows the embedding of the nodes where vehicles are parking or are heading to. We append to each row two values: the first one is the current capacity of the vehicle and the second one is the consumed weight (distance) till now.

$$CurrentNode \quad Requests \quad Vehicles$$

$$\begin{pmatrix} e_{11} & e_{12} & \cdots & e_{1m} \end{pmatrix} \quad \begin{pmatrix} e_{11} & e_{12} & \cdots & e_{1m} & q_1 \\ e_{21} & e_{22} & \cdots & e_{2m} & q_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ e_{n1} & e_{n2} & \cdots & e_{nm} & q_n \end{pmatrix} \quad \begin{pmatrix} e_{11} & e_{12} & \cdots & e_{1m} & c_1 & w_1 \\ e_{21} & e_{22} & \cdots & e_{2m} & c_2 & w_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \\ e_{v1} & e_{v2} & \cdots & e_{vm} & c_v & w_v \end{pmatrix}$$

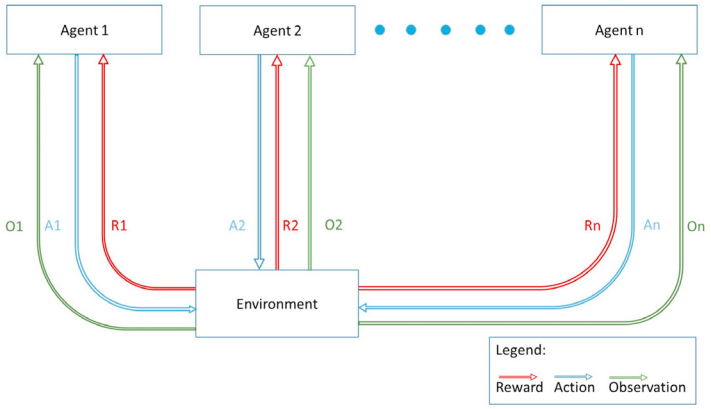

Fig. 2. Decentralized Training for Decentralized Control (DTDC) paradigm

4.3. Agents

Generally, DS-VRP and other versions of VRP have a number of nodes greater than the number of vehicles serving customers placed in these nodes. Unlike many previous works, in this one, we decided to put RL agents in graph nodes instead of the vehicles. This means that we will have many more RL agents than others, but at the same time, we now have the possibility to make our neural nets smaller in terms of the number of parameters. The direct benefit of this method lies in the possibility of obtaining a good solution but with a relatively lower number of parameters that leads to fast inference time and hopefully fast training procedure.

Building models in vehicles exposes them to encounter all the different types of typologies presented in the graph, which will make it harder for models to learn due to the difficulty and diversity of the patterns. On the contrary, placing RL agents in nodes will reduce the size of the private space for each agent and thus a greater possibility of learning and making more effective decisions. Now, the model has a specific number of neighbors, and it is familiar with the weights that separate it from them and the profit and loss resulting from any decision it makes based on the vehicle configuration.

There are multiple paradigms for training and running multi-agent reinforcement learning agents. Decentralized Training for Decentralized Control (DTDC) paradigm [8] also called Independent Learners is the most famous one where each agent tries to get the highest possible cumulative rewards not taking into consideration other agents' actions, see Fig. **??**. While the independent learners have to build individual statistics to drive their optimization, the extra information we have available during training in CTDC makes it possible to maintain a joint statistic that helps avoid local minima and can lead the agents to coordinate themselves better when executing their individual policies in a decentralized way. Also, given that we have found a solution for a certain size of a graph, we can simply scale it to any type of vehicle in use. Usually, in real-world problems, the number of vehicles in use changes dramatically more often than the size of the graph. In this work, we experiment with how DTDC can help in solving DS-VRP.

4.4. Reinforcement learning for agents

In this paper, we experimented with two types of reinforcement learning algorithms: value-based and policy-based.

DQN-based agent: For value-based approach, we used Deep Q-learning (DQN) algorithm [18]. Our agents here are simple feed-forward neural network models with 5 hidden layers with sizes: [256, 256, 512, 256, 128]. To train the agent, we built different scenarios with the same graph size. Then we evaluated the mean and the standard deviation of the cumulative reward function that our agents get at the end of each episode.

PPO-based agent: Proximal policy optimization (PPO) [21] is a policy gradient method and it achieved very well results on different tasks. The only difference between this experiment and the previous one is the core algorithm and all other details about scenario generating and result calculation are the same.

The output of the previous models was the Q-value function which approximated the estimated cost of serving the current requests going through a predefined neighbor. Here we are trying to predict the GLEE embedding of the next node that we should go through. And our loss function will be the Euclidean distance between the prediction and the real value of the embedding. This will help our models in the future to generate other typologies using FastGe [2] algorithm, for example. We depend on actor-critic architecture where our actor will predict the embedding of the next node and our critic will predict the Q-value of this choice.

Two phases of learning: When we start training these agents, we noticed how hard for a neural network model it was to converge and learn to route the vehicles in the correct directions, so we started to free up some restrictions to see what are the problem exactly. Indeed, the existence of many restrictions at the same time made it hard for our agents to learn.

This is why we split our training phase into two phases:

First Phase: we train our agent with no capacity or weight restriction. In other words, we train the agents just to handle the weight of the edges in the graph and to be as good as possible with routing the vehicles to serve the real-time requests by using the edges with the minimum weights. This can be considered as a pretraining step.

Second Phase: we add back our capacity and weight restrictions limit and continue training.

5. Experiments and Results

5.1. GLEE + DQN

We came up with the following configuration of DQN. Learning configurations are: learning_rate : 0.0001, batch_size : 64, mem_capacity : 10000, activation function: relu, optimizer: RMSProp, embedding_dim : 8, epsilon_decay: 0.9999.

For the first phase, we trained agents for 200,000 requests with new requests every [2,5] random time unit. For the second phase, We trained agents in an episodic manner for 500 episodes. Each episode is for 20 requests with a 150 capacity and 200 weight

We noticed as you can see in Figure 3 how our model managed to minimize the average distance or weight consumed to serve new requests. Also We can see in Figure 3 how the agent manages to decrease also the average time needed to sever new requests but this time with all the restrictions in place. An important note we need to mention is that in both phases we kept some exploration ratio around 20%, to help the models constantly update their Q-values models. We ran experiments on different graph sizes to test the scalability and stability of our approach. We also recorded the run time of training on one instance scenario of the problem for each graph size.

Table 1. Run time of different graph sizes of the first and second phase of training.

Graph size N	First Phase run time (hh:mm)	Second Phase run time (hh:mm)
10	01:20	00:16
20	05:30	00:23
50	23:02	02:11

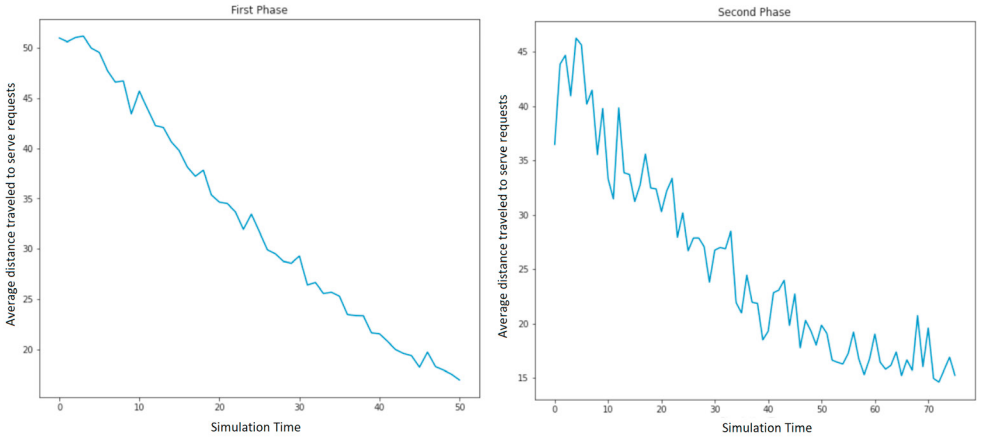

Fig. 3. GLEE + DQN — First (left) and second (right) training phase

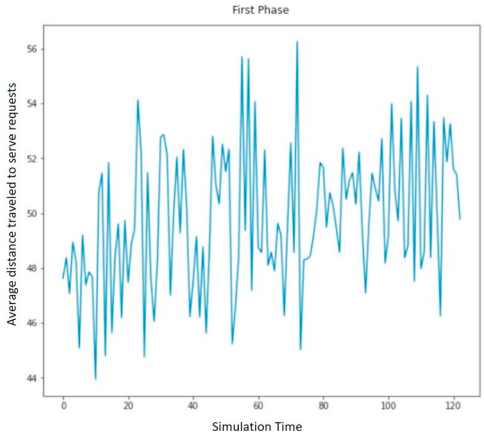

Fig. 4. GLEE + PPO — First training phase

5.2. GLEE + PPO

After conducting many experiments with different model sizes, architectures, different optimization methods, and hyper-parameters we failed in training these models to do the task and converge. Figure 4 shows how models are failing in converging as the average distance (weight) consumed by vehicles to serve requests increase with time.

5.3. Comparison with baselines

As the baselines, we used two methods:

- classical Google OR-tools that implements local search. It is important to understand about this algorithms that it has to be run for every new instance, which is opposite to anything you can call “adaptive”. If you freeze the weights in DQN, only inference would take seconds, which is not an option for OR-tools that would have to find a new solution each new time. So we took it as a state-of-the-art classical non-adaptive algorithm to see if we can beat its performance
- Transformer-based MARDAM model [9], which was overviewed in Section 2. This is an adaptive model.

Table 2 shows the comparison of algorithm performance. As you can see, the models shows similar enough performance in terms of time taken to solve each instance (a specific graph) from a scratch.

Table 2. Comparison of the approaches, N is the graph size.

Method	$N = 10$	$N = 20$	$N = 50$
OR Tools	3.41 ± 0.51	5.39 ± 0.56	10.64 ± 1.16
MARDAM	3.38 ± 0.47	5.46 ± 0.57	10.88 ± 1.20
GLEE + DQN	3.40 ± 0.80	5.43 ± 0.77	11.34 ± 1.52
GLEE + PPO	∞	∞	∞

What makes the difference, our solution is relatively small. While MARDAM contains 692k trainable parameters. In our case, we place a DQN-model in each node, and size of each DQN depends on the input size. For the largest case, $N = 50$, all the DQN-model used contained 360k trainable parameters. What is even more important, our multi-agent solution is also adatpive for the scenario of graph change. Utilizing approaches such as [2] would help to avoid retrainig the pipeline for the graph change, which is not applicable for the MARDAM model.

6. Conclusion

In this paper, we presented a novel method to handle DS-VRP. Our approach depends on multi-agent reinforcement learning where we place our decision makers in nodes instead of vehicles and we use geometric Laplacian eigenmaps embedding to represent graph nodes information. We also presented a new method of training RL agents in a graph by separating it into two phases, the first one with all VRP constraints removed and the second one with them added back. Also, we built a simulation to develop and test different methods for the DS-VRP problem. We tested our new method combined with two RL algorithms, a value-based one and a policy-based one. DQN gives relatively good results while PPO failed to converge.

The results showed that the DQN-based method shows the same performance being trained from the scratch as Google OR tools and MARDAM, while being more adaptive than these methods and having less parameters than MARDAM. We can conclude that the proposed approach is more efficient in solving the most complicated routing problem statement in comparison with the state-of-the-art algorithms.

The proposed approach can receive further developments and may show even better results with future enhancement and have a great potential in solving other distributed routing problems, especially in the logistic domain where similar tasks can be required to solve. Another direction of further research is providing a theoretical basis explaining the method and development of verification algorithms to investigate its robustness.

Acknowledgements

The work was financially supported by the Russian Science Foundation (Project 20-19-00700).

References

[1] Baker, B.M., Ayechew, M., 2003. A genetic algorithm for the vehicle routing problem. Computers & Operations Research 30, 787–800.
[2] Barbahan, I., Baikalov, V., Vyatkin, V., Filchenkov, A., 2021. Multi-agent deep reinforcement learning-based algorithm for fast generalization on routing problems. Procedia Computer Science 193, 228–238.
[3] Belkin, M., Niyogi, P., 2003. Laplacian eigenmaps for dimensionality reduction and data representation. Neural computation 15, 1373–1396.
[4] Bell, J.E., McMullen, P.R., 2004. Ant colony optimization techniques for the vehicle routing problem. Advanced engineering informatics 18, 41–48.
[5] Bello, I., Pham, H., Le, Q.V., Norouzi, M., Bengio, S., 2016. Neural combinatorial optimization with reinforcement learning. arXiv preprint arXiv:1611.09940 .
[6] Bengio, Y., Lodi, A., Prouvost, A., 2021. Machine learning for combinatorial optimization: a methodological tour d'horizon. European Journal of Operational Research 290, 405–421.

[7] Bent, R.W., Van Hentenryck, P., 2004. Scenario-based planning for partially dynamic vehicle routing with stochastic customers. Operations Research 52, 977–987.
[8] Bono, G., Dibangoye, J.S., Matignon, L., Pereyron, F., Simonin, O., 2018. Cooperative multi-agent policy gradient, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer. pp. 459–476.
[9] Bono, G., Dibangoye, J.S., Simonin, O., Matignon, L., Pereyron, F., 2020. Solving multi-agent routing problems using deep attention mechanisms. IEEE Transactions on Intelligent Transportation Systems 22, 7804–7813.
[10] Cordeau, J.F., 2006. A branch-and-cut algorithm for the dial-a-ride problem. Operations Research 54, 573–586.
[11] Dantzig, G.B., Ramser, J.H., 1959. The truck dispatching problem. Management science 6, 80–91.
[12] Desaulniers, G., Desrosiers, J., Erdmann, A., Solomon, M.M., Soumis, F., 2002. Vrp with pickup and delivery. The vehicle routing problem 9, 225–242.
[13] Deudon, M., Cournut, P., Lacoste, A., Adulyasak, Y., Rousseau, L.M., 2018. Learning heuristics for the tsp by policy gradient, in: International conference on the integration of constraint programming, artificial intelligence, and operations research, Springer. pp. 170–181.
[14] Hemmelmayr, V.C., Cordeau, J.F., Crainic, T.G., 2012. An adaptive large neighborhood search heuristic for two-echelon vehicle routing problems arising in city logistics. Computers & operations research 39, 3215–3228.
[15] Kool, W., Van Hoof, H., Welling, M., 2018. Attention, learn to solve routing problems! arXiv preprint arXiv:1803.08475 .
[16] Mazyavkina, N., Sviridov, S., Ivanov, S., Burnaev, E., 2021. Reinforcement learning for combinatorial optimization: A survey. Computers & Operations Research 134, 105400.
[17] Meng, X., Li, Z., Tang, J., 2021. A dynamic vrp with varying transportation costs and its solution strategy, in: 2021 IEEE International Conference on Networking, Sensing and Control (ICNSC), IEEE. pp. 1–6.
[18] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M., 2013. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 .
[19] Pillac, V., Gendreau, M., Guéret, C., Medaglia, A.L., 2013. A review of dynamic vehicle routing problems. European Journal of Operational Research 225, 1–11.
[20] Saint-Guillain, M., Deville, Y., Solnon, C., 2015. A multistage stochastic programming approach to the dynamic and stochastic vrptw, in: International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research, Springer. pp. 357–374.
[21] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., 2017. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 .
[22] Secomandi, N., Margot, F., 2009. Reoptimization approaches for the vehicle-routing problem with stochastic demands. Operations research 57, 214–230.
[23] Torres, L., Chan, K.S., Eliassi-Rad, T., 2020. Glee: Geometric laplacian eigenmap embedding. Journal of Complex Networks 8, cnaa007.
[24] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I., 2017. Attention is all you need. Advances in neural information processing systems 30.
[25] Vinyals, O., Fortunato, M., Jaitly, N., 2015. Pointer networks. Advances in neural information processing systems 28.