IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 30 September 2023, accepted 18 October 2023, date of publication 25 October 2023, date of current version 6 November 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3327324

RESEARCH ARTICLE

A Novel Routing Solution for EV Fleets: A Real-World Case Study Leveraging Double DQNs and Graph-Structured Data to Solve the EVRPTW Problem

DIMETH NOUICER[1,2], **IKBAL CHAMMAKHI MSADAA**[1], **AND KHALED GRAYAA**[1]
[1]LaRINa, ENSTAB, University of Carthage, Borj Cédria 1054, Tunisia
[2]ENSIT, University of Tunis, Tunis 1008, Tunisia

Corresponding author: Dimeth Nouicer (dimeth.nouicer@ensit.u-tunis.tn)

This work was supported by the German Academic Exchange Service (DAAD), Federal Ministry for Economic Cooperation and Development (BMZ), Germany, within the framework of the Renewable Energy-based E-Mobility in Higher Education (REMO) Project under Grant 57545562.

ABSTRACT The transition from Internal Combustion Engine vehicles to Electric Vehicles presents challenges for fleet managers in terms of adapting operational processes and systems. One prominent challenge is the Electric Vehicle Routing Problem, which requires careful consideration of battery monitoring, efficient route planning, charging infrastructure availability, and vehicle performance management. The goal is to alleviate range anxiety and ensure effective fleet management. In this context, we propose a Reinforcement Learning approach in conjunction with graph-based modeling to solve Electric Vehicle Routing Problem with Time Window. This paper provides a novel approach addressing the need for efficient and sustainable electric vehicle fleet management. Our aim is to minimize the distance traveled while serving customers within their time windows using the combination of Structure2vect and Double Deep Q-Network. Real-world data from a public utility fleet company in Tunisia is utilized to evaluate the proposed model, and comparisons are made with conventional benchmarking strategies and other Reinforcement Learning approaches. The results highlight the effectiveness of the proposed model achieving a reduction up to 50% in traveled distance while demonstrating enhanced computational efficiency through reduced complexity and optimized runtime. Moreover, the obtained model can be directly applied to treat the large-scale adoption of electric vehicles in fleets.

INDEX TERMS Deep reinforcement learning, DDQN, electric vehicles, EVRPTW, fleet, graph structure.

I. INTRODUCTION

Transitioning from Internal Combustion Engine (ICE) vehicles to Electric Vehicles (EVs) requires fleet managers to adapt their operational processes and systems while carefully assessing their daily mileage requirements to anticipate the issue of range anxiety. Monitoring battery levels, planning efficient routes that consider the availability and accessibility of charging infrastructure, optimizing charging schedules, and managing vehicle performance data become critical for effective fleet management. In this context, the Electric Vehicle Routing Problem (EVRP) emerges as a prominent challenge that requires attention for the successful large-scale adoption of EVs in fleets. The Vehicle Routing Problem (VRP) is an NP-hard combinatorial optimization (CO) problem that involves finding the optimal route for a fleet of vehicles to ensure deliveries or services for a set of customers in different locations. It is a generalization of the classical Traveling Salesman Problem (TSP), where only one vehicle serves multiple customers. The VRP has various variants

The associate editor coordinating the review of this manuscript and approving it for publication was Miadreza Shafie-Khah.

based on different constraints and considerations: Capacited VRP (CVRP) where the vehicle has a limited capacity, VRP with time window (VRPTW) [1], which introduces the time window within which the customer needs to be serviced, etc. In this paper, we focus on CEVRPTW: the electric vehicle-oriented variant of VRP with time window and capacited vehicles constraints. Like for many other CO problems, Reinforcement Learning (RL) can be an appealing choice to solve the EVRP problem. Once formulated as a Markov Decision Process (MDP), the EVRP becomes a candidate for an RL solution, allowing an agent—in this case, the EV fleet—to learn in an interactive environment through trial and error, relying on evaluative feedback that depends on its own actions and past behavior [2]. Given the lack of prior knowledge about the environment dynamics, a model-free reinforcement learning algorithm based on Q-learning is well-suited to our case. Specifically, we utilize the Double Deep Q-Network (DDQN), which combines reinforcement learning with deep learning techniques to understand the underlying complexity of the environment and generalize to new, unseen situations. Another important aspect of our solution is the incorporation of graph structured information. Indeed, routes optimization problems are fundamentally represented as graphs. Routes represent the edges of a graph that link the customers, the fleet depot and the charging stations, which all represent the nodes of the input graph. This graph-based representation allows for efficient navigation and optimization techniques to be applied and can lead to more accurate and realistic modeling of the EVRP.

In this paper, we aim to explore the potential synergy between DDQN and graph-based modeling. DDQN can be used to learn efficient routing policies by considering the graph structure of the problem. Based on our current investigation of the field, this is the first time that Double DQN is considered to solve EVRPTW problem while keeping the graph-based representation of the problem using strcuture2vect (S2V). We demonstrate that our model outperforms state-of-the-art solutions while using a less complex architecture. Furthermore, we show its effectiveness with both synthetic and real-world data.

The novelty of this paper lies in the development of an innovative and efficient approach to tackling the EVRPTW. For the first time, we harness the synergy between DDQN and graph-based modeling to solve this problem, employing well-defined reward structures and finely-tuned hyper-parameters. This work not only breaks a new ground in the field but also offers a practical solution with reduced complexity, improved performance in terms of runtime and traveled distance, making it a valuable contribution to the existing body of research.

The remainder of this paper is organized as follows. In Section II, we discuss related articles presenting reinforcement learning models that solve EVRP and TSP variants. In Section III, we define the problem formulation and MDP model. In Section IV, we present our model using the combination of DQN and S2V. In Section V, we present a comprehensive set of experiments with both synthetic and real world data scenarios. Finally, we draw our conclusions and point out future work directions in Section VI.

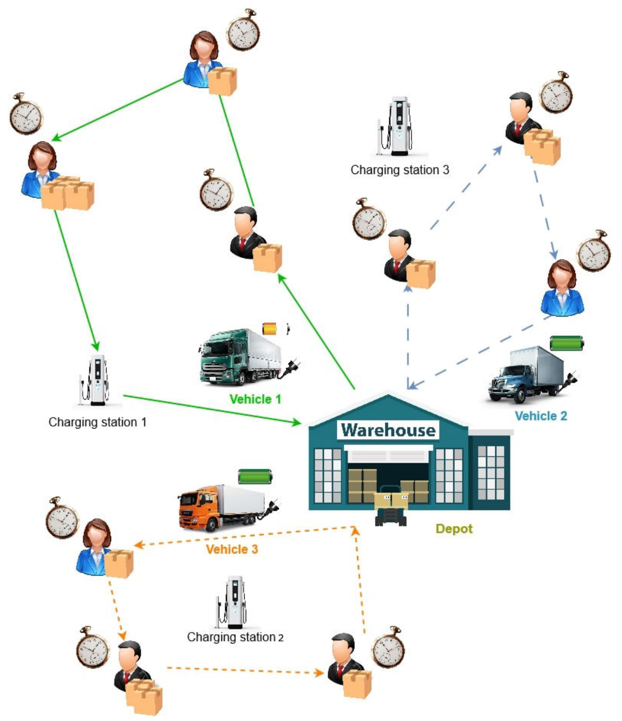


FIGURE 1. EVRPTW problem.

II. RELATED WORK

Over the past few years, significant efforts have been made to address the environmental concerns associated with EVs, particularly in reducing energy consumption. Consequently, the Electric Vehicle Traveling Salesman Problem (EVTSP) has emerged as a crucial focus area. The EVTSP aims to minimize both the total distance traveled and the energy consumed while considering the need for EVs to recharge their batteries at charging stations (CS) along the route. Various approaches have been proposed to address the Electric Vehicle Traveling Salesman Problem (EVTSP). In terms of charging navigation strategy, Basso et al. [4] employed Full Bayesian Regression, while Aygun et al. [3] utilized heuristics combining Dijkstra and Floyd-Warshall algorithms. Reinforcement learning techniques, which are the main focus of the current study, have also been explored. The Q-learning algorithm was applied by Ottoni et al. [6] and Dorokhova et al. [5] while the authors of [9] used the State-Action-Reward-State-Action (SARSA) algorithm. Similarly [6] used SARSA to compare with Q learning. Deep Reinforcement Learning (DRL) algorithms such as Double Deep Q-Network (DDQN) [8] and Dueling Deep Q-Network (DQN) [7] have also been explored. The Dueling DQN was specifically used to optimize both travel and energy costs. Jin and Xu [16] investigated the charging routing problem for fleets and proposed an Actor-Critic (A2C) solution to minimize the charging cost and the sum of travel costs

TABLE 1. Summary of different approaches.

Reference	Problem	ML algorithm	Graph	Demand	TW	Data type
Aygun et al. [3]	EVTSP	Dijkstra and Floyd-Warshall	✗	-	-	real world
Basso et al. [4]	EVTSP	Full Bayesian Regression	✗	-	-	real world
Dorokhova et al. [5]	EVTSP	Q learning	✗	-	-	real world
Ottoni et al. [6]	EVTSP	Q learning, SARSA	✗	-	-	real world
Zhang et al. [7]	EVTSP	DQN	✗	-	-	synthetic/real world
Aljohani et al. [8]	EVTSP	Double DQN	-	-	✗	real world
Aljohani et al. [9]	EVTSP	SARSA	✗	-	-	real world
Xiang et al. [10]	EVTSP	Rainbow DQN	GAT	-	-	real world
Turan et al. [11]	Autonomous CEVRP	PPO	✗	dynamic	✓	real data
Basso et al. [12]	CEVRP	Q learning	✗	dynamic	✗	real world
Lin et al. [13]	CEVRPTW	Reinforce	GAT, S2V	static	✓	synthetic
Qiu et al. [14]	CEVRPTW	A2C, BS	M-H GAT	static	✓	real world
Tang et al. [15]	CEVRPTW	A2C	GAT	static	✓	real world
Current work	**CEVRPTW**	**Double DQN**	**S2V**	**static**	✓	**synthetic/real world**

to charging stations. The same algorithm was utilized by Bogyrbayeva et al. in [17] combined with Graph Attention Mechanism (GAT) to solve the EVTSP.

Unlike the EVTSP, the EVRP assumes the existence of multiple EVs (not just one), customers with static or dynamic demands, charging stations and a depot as illustrated in Figure 1. The vehicles must satisfy the customers before the end of the planning horizon while minimizing the total traveled distance. The EVRP may also involve various objectives such as maximizing resource utilization or minimizing costs related to travel time and energy consumption. When the vehicle has a limited capacity, it needs to return to the depot during the day to load more goods and be able to serve the customers: this variant is known as Capacited EVRP (CEVRP). In this context, a recent study conducted by Basso et al. [12] employed safe RL with Q-learning to address the CEVRP with stochastic demands. The study paid careful attention to monitoring the battery State of Charge (SoC) to ensure that energy constraints are not violated during the optimization process. A variant of the EVRP that incorporates time constraints is known as the EVRP with Time Window (EVRPTW), where customers must be served within specific time windows. In this context, several approaches have utilized RL combined with Graph Neural Networks (GNN) to represent the problem as a graph. The initial introduction of this approach was made by [18] and [19].

To address the EVRPTW, Lin et al. [13] proposed a REINFORCE model augmented with Graph Attention Mechanism (GAT) and Structure2vec (S2V) [20], enabling the synthesis of local and global information from the graph representation. Their objective was to serve all customers within their time windows while minimizing the total distance traveled by the fleet. Qiu et al. [14] proposed an A2C algorithm employing a Multi-head Attention (MH ATT) encoder for state feature extraction, along with a Beam Search-based decoder. Tang et al. [15] combined A2C with GAT to solve the EVRPTW. Notably, their work introduced an energy consumption model that considered factors such as EV mass, speed, acceleration, and road gradient. The authors of [10] utilized a dual-agent architecture consisting of a GAT and a Rainbow DQN, an alternative DQN architecture.

Table 1 provides a summary of the studies that have explored the EVTSP and EVRP problems utilizing Machine Learning or Reinforcement Learning methodologies. The table highlights that, to the best of the authors' knowledge, there has been limited utilization of graph-structured information in RL methods for solving the EVRP.

In this paper, our contribution lies in the application of DDQN with graph embeddings and Structure2vec to solve CEVRPTW. This combination of techniques, to the best of our knowledge, has not been previously explored for this specific problem. Additionally, we test the performance of our model by utilizing real-world data, to further increase the relevance and applicability of our findings.

III. PROBLEM STATEMENT

In this paper we consider Capacitated Electric Vehicle Routing Problem with Time Windows but, for the sake of simplicity, in the rest of this paper we refer to it as EVRPTW. The problem is modeled as a fully connected graph $G = (V, A)$ with 3 types of Vertices: customers (Vc), charging stations (Vs) and a depot (Vd) as illustrated in Figure 1. We consider having only one single depot which is always denoted by the node 0. Each edge $(i, j) \in E$ in the graph is associated with the distance $d_{i,j}$, the energy cost $e_{i,j}$ and the travel time $\Omega_{i,j}$. The goal is to minimize the total distance traveled by all the EVs in the fleet while serving all the customers within a certain time window.

Each vertex i is associated with an array $X_i^t = (x_i, y_i, s_i, e_i, d_i^t)$ where (x_i, y_i) denotes the coordinates of a vertex i. The pair (s_i, e_i) indicate the starting and ending time of the time window associated with each vertex. A time window specifies the allowable time interval during which the customer must be served. Additionally, d_i^t represents the demand at vertex i at step t. For the depot and CS demand is always 0, and if a customer $i \in Vc$ is visited its demand is set to 0. As we are normalizing all the variables, the time window of the CSs and the depot is set as [0,1] where 1 represents the end of the planning horizon.

TABLE 2. Variables and parameters definitions of the EVRPTW model.

Symbol	Definition
T	the end of the planning horizon
t	current time step
V	set of vertices
Vc	set of customers vertices
Vs	set of charging stations vertices
Vd	depot
E	set of edges
$d_{i,j}$	distance between node i and j
$e_{i,j}$	energy cost between node i and j
$\Omega_{i,j}$	travel time between node i and j
$\Omega_{v^t,v^{t+1}}$	travel time between v^t and v^{t+1}
x_i	coordinate of node i in X-axis
y_i	coordinate of node i in Y-axis
s_i	start time window of node i
e_i	end time window of node i
d_i^t	demand of node i at step t
$load_n^t$	load of EV n at step t
$load_n^{t_0}$	load of EV n at step t_0 corresponding to maximum load of EV n
soc_n^t	battery state of charge of EV n at step t
T^t	system time at step t
N^{EV}	total number of EVs in the fleet
$V^T = \{i, j, ..., k\}$	set of all visited vertices, corresponding to final solution sequence
$V^t = \{i, j, ..., k\}$	set of visited vertices at step t
v^t	last visited node at step t
Ω_s	service time at each node $i \in Vc$
Ω_c	charging time in charging station

We also keep track of EV characteristics ($load_n^t$, soc_n^t), where at each step t, $load_n^t$ denotes the load of EV n, and soc_n^t represents the EV n battery state of charge (SoC). In this paper, we assume a fleet of homogeneous EVs with the same maximum capacity denoted by $load_n^{t_0}$ which is the load of EV n at the start of the planning horizon t_0. As a result, the EV load cannot exceed the EV n maximum capacity: $load_n^t \leq load_n^{t_0}$. Finally, we consider global variables T^t and N^{EV} where T^t represents the system time and N^{EV} indicates the number of EVs in the fleet.

The solution is a sequence of visited vertices denoted as $V^T = \{i, j, \ldots, k\}$ where V^T represents the set of all the visited vertices with different EV routes separated by the depot. It is important to note that every solution starts and ends at the depot. If we consider this solution example $\{0, 1, 2, 3, 0, 4, 5, 6, 0\}$: this sequence implies that there are 2 EVs used, one EV route is $\{0, 1, 2, 3, 0\}$ and the second EV route is $\{0, 4, 5, 6, 0\}$. Table 2 summarizes the variables and parameters discussed above.

IV. METHODOLOGY

In this section, we describe the model we use to solve EVRPTW as stated before. Our solution consists of combining graph-structured data with DRL.

A. GRAPH EMBEDDINGS

Graphs are an effective structure for representing network data, allowing us to capture connections between customers, charging stations (CS), and road topology features. Unlike traditional data structures such as arrays or lists, graphs lack built-in tools and structures compatible with ML algorithms. To address this challenge, graph embeddings were introduced, where the whole graph is encoded in one single vector like a compressed representation. In contrast, using an adjacency matrix as a direct representation for large graphs, such as those with 100 nodes, becomes highly impractical due to the resulting matrix size (e.g., a 100 × 100 matrix), which leads to significant computational costs and excessive memory requirements [21].

In this paper, we use S2V [20] to represent the EVRPTW graph in an embedded feature space. S2V extracts graph features by performing a sequence of nonlinear function mappings as represented in Equation 1 [20]. We use this representation to extract local and global information of the graph state and feed it to the DDQN model. The model input X_i^t is first embedded into $\hat{X}_i^t$ vector using a linear layer. It is then used in S2V to generate the graph embedded vector μ_i^k.

$$\mu_i^k = relu(\theta_1 \hat{X}_i^t + \theta_2 \sum_{j \in N_i} \mu_j^{k-1} + \theta_3 \sum_{j \in N_i} relu[\theta_4 d_{i,j}]). \quad (1)$$

In equation 1 [20], N_i is the set of connected vertices (neighboring nodes that are connected by an arc), $d_{i,j}$ is the distance between vertex i and j. θ_1, θ_2, θ_3 and θ_4 are trainable variables and $relu$ is a non-linear activation function. First, μ_i^k, where $k \in [0, .., p]$, is initialized with $\mu_i^0 = \hat{X}_i^t$ and then updated recursively for p rounds following equation 1 to generate the final μ_i^p for each vertex i. Afterwards, the final vector μ_i^p will be the graph representation and the input to the RL model we use to solve EVRPTW.

B. REINFORCEMENT LEARNING

The RL method of the EVRPTW comes with five major components: the agent, the environment, the state, the action and the reward. The agent is the decision maker, which takes action based on the environment situation called state at decoding step t. The environment will give a reward to the agent depending on how good his action is. Over the time, the agent will try to maximize the cumulative reward [2]. The agent learns a policy which is a mapping between the tuples (states, actions). A good policy is when it serves to optimize the expected sum of rewards.

To solve this problem, we need to transform it into an MPD formulation, where the environment represents the graph $G = (V, E)$ described above with the different vertices and edges. Inspired by the work of Bdeir et al. [22], we define the state space with some modifications. At each step t, the state is represented by a combination of vertices coordinates and boolean variables. Specifically, the state is defined as $S_t = (\text{coords}, \sigma_i^t, \lambda_i^t, \delta_i^t)$ at step t where coords are the vertices coordinates, σ_i^t denotes whether a vertex i is already in solution or not, λ_i^t represents if vertex i is the last visited or not (i.e., corresponds to the EV current position) and δ_i^t represents whether the demand of vertex i is null (i.e., satisfied). If V^t represents the set of vertices visited at step t and v^t represents the last visited node at step t, Equations (2) - (4) represent the boolean variables of the state space.

$$\sigma_i^t = \begin{cases} 1 & \text{if } i \in V^t \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

$$\lambda_i^t = \begin{cases} 1 & \text{if } i = v^t \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

$$\delta_i^t = \begin{cases} 1 & \text{if } d_i^t > 0 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

In RL models, the agent will take the decision based on the state of the environment as mentioned earlier. For this reason, the RL state will be modeled as a graph and will serve as the input for the graph embeddings in the S2V model, specifically as the vector X_i^t referenced in Equation 1.

The action space represents a set of discrete actions $a(a \in A) := (Vc, Vs, Vd)$. The agent decides to go to a customer (Vc), a station (Vs) or a depot (Vd).

At each step t, we update the state based on the transition functions (5) - (8). First, we update the customers remaining demands and EV load as follows.

$$d_i^t = \begin{cases} 0 & \text{if } i \in V^t \\ d_i^{t-1} & \text{otherwise} \end{cases} \tag{5}$$

$$load_n^t = \begin{cases} load_n^{t-1} - d_i^{t-1} & \text{if } i \notin Vd \\ & \text{and } i = v^t \\ load_n^{t_0} & \text{if } i \in Vd \end{cases} \tag{6}$$

Second, the EV SoC is updated as follows.

$$soc_n^t = \begin{cases} soc_n^{t-1} - soc(d_{i,j}) & \text{where i} = v^{t-1} \text{and} j = v^t \\ soc_n^{t_0} & \text{otherwise} \end{cases} \tag{7}$$

Finally, we update the system time T in Equation 8. If v^t represents the last visited vertex at step t, $\Omega_{v^t,v^{t+1}}$ is the traveling time between v^t and v^{t+1}. Ω_s represents the service time at each node $i \in Vc$. To simplify, we assume the service time to be the same constant for all customers. The Ω_c is the charging time in a charging station. In this paper, we assume a linear charging time like in [13].

$$T^t = \begin{cases} T^{t-1} + \Omega_{v^{t-1},v^t} + \Omega_s & \text{if } v^t \in Vc \\ T^{t-1} + \Omega_{v^{t-1},v^t} + \Omega_c & \text{if } v^t \in Vs \\ 0 & \text{if } v^{t+1} \in Vd \end{cases} \tag{8}$$

Regarding the reward, after experimenting with various reward formulations, we have opted for one that is inversely proportional to the distance traveled as represented in Equation 9. Unlike already existing works [13], [15], [18], [19], [23] who mostly adopt a negative distance, this formulation gives better results. In reinforcement learning, the updates of Q-values are directly influenced by the magnitude of rewards. Our observations have shown that when reward variations are insufficiently large, the model struggles to adapt effectively. To address this issue, we employ a reward formulation that is inversely proportional to the traveled distance, as it introduces higher variance compared to a negative distance-based reward. This reward formulation is like a penalty based on the distance and the action chosen by the agent as represented in Equation 9. The longer the distance, the more the agent is penalized. Additionally, if the number of EVs used (N_{used}^{EV}) at time step t in the solution exceeds the total number of available EVs in the fleet (N^{EV}), there is an additional penalty for the distances traveled by the extra EVs in their respective routes, as specified in Equation 9.

$$r^t = \begin{cases} \dfrac{1}{d_{v^t,v^{t+1}}} & \text{if } N_{used}^{EV} \leq N^{EV} \\ \dfrac{1}{2 * d_{v^t,v^{t+1}}} & \text{if } N_{used}^{EV} > N^{EV} \end{cases} \tag{9}$$

For simple environments, there are several RL methods to use. However in our case, the environment is high-dimensional and complex as it depends on several variables as explained above. As a result, model-free DRL algorithms are more adapted specifically DQN.

C. DQN

DQN is an off-policy model-free algorithm used to solve problems with discrete actions. The architecture we use is illustrated in Figure 2. DQN is a value-based algorithm that attributes a Q-value to actions based on the state s at step t and the agent will pick the action a with the maximum Q(s,a). This Q-value is an estimation of how good a certain action is at the given state, and it is updated following the Bellman equation 10 [2].

$$\begin{aligned} Q(s_t, a_t) = (1 - \alpha).Q(s_t, a_t) + \alpha.[r_t(s_t, a_t) \\ + \gamma.max(Q(s_{t+1}, a_{t+1})] \end{aligned} \tag{10}$$

The model takes the environment state s as an input and outputs Q-values $Q(s_t, a_t)$ of each possible action a at step t based on the learning rate α, the reward $r_t(s_t, a_t)$ at timestep t, the discount factor γ, which is responsible for prioritizing either immediate or future rewards (depending on the value), and the Q value of the next state and next action $Q(s_{t+1}, a_{t+1})$. After training, the model will be able to approximate Q-values for actions that maximize the cumulative reward. However, an instability can be caused in this architecture when calculating $Q(s_t, a_t)$. In the Bellman equation (10), $Q(s_t, a_t)$ is computed based on $Q(s_{t+1}, a_{t+1})$, where s_{t+1} is only one step ahead of s_t. Both states are determined using the same Q network, resulting in a situation of pursuing a moving target [2]. To address this issue, a second neural network called the Target network is introduced to estimate $Q(s_{t+1}, a_{t+1})$. This approach, known as Double Deep Q-Network (DDQN), utilizes two neural networks for estimating Q-values.

The training steps are illustrated in Figure 2 and the pseudo code is summarized in Algorithm 1. Each training episode simulates a complete tour of a single graph from depot,

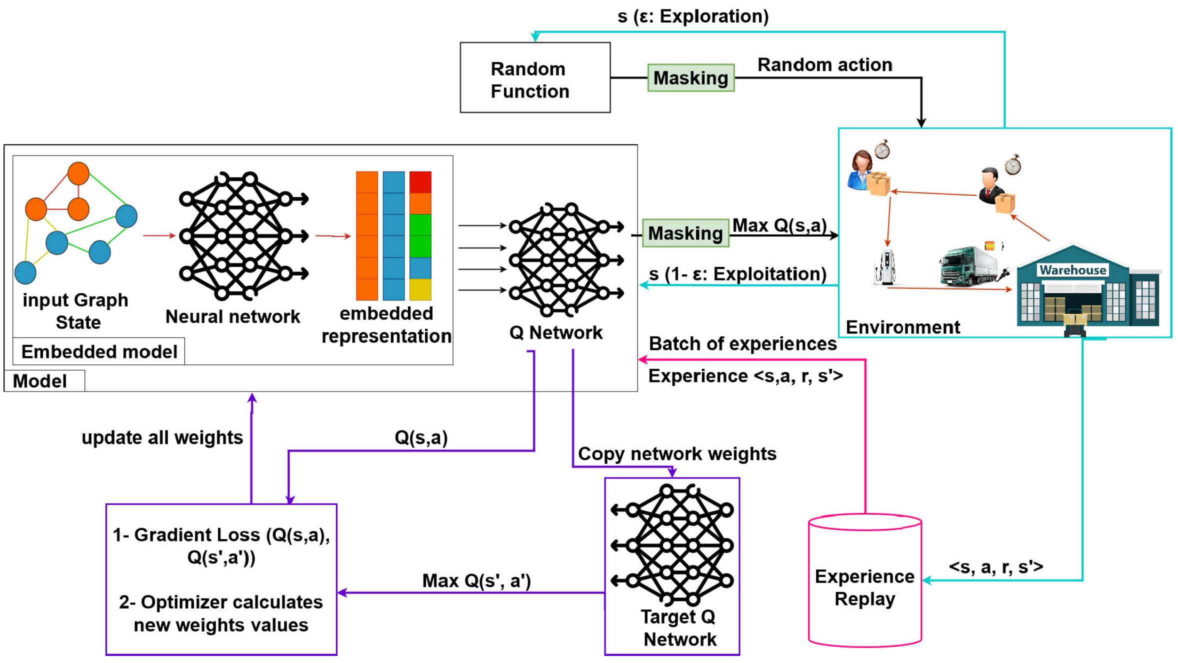


FIGURE 2. **The proposed model architecture with DQN training method.**

to serving all customers to going back to the depot at the end. The initial step involves the initialization of the neural network weights. In each subsequent training iteration, a new environment is initialized, generating the initial state s_0. The reward is initially set to zero, and an experience replay memory is created to store the agent's experiences. Following this, a while loop is executed until the agent reaches the final state, which occurs either when the tour ends and all the customers are served, or when the agent reaches the end of the planned horizon.

At each new state s_t, the model selects an action based on exploration or exploitation strategy using an ϵ-greedy policy. Exploration occurs when the agent decides to explore the environment and randomly chooses an action. Exploitation involves exploiting the model by computing the Q-values for all possible actions, and selecting the action having the maximum Q-value $MaxQ(s_t, a_t)$ based on the greedy approach. Subsequently, the chosen action is applied to the environment, resulting in a reward and the next state s_{t+1} denoted as s' in Figure 2.

These experiences are stored in the experience replay memory and utilized for model training at a later stage. When a sufficient number of experiences are accumulated in the replay memory, a batch of <states, actions, rewards, next states>, denoted as $< s, a, r, s' >$, is sent to the model to compute the loss. During this step, the model recalculates $Q(s_t, a_t)$ for each state-action pair in the batch, while the target network computes $Q(s_{t+1}, a_{t+1})$ for each next state s_{t+1} considering all possible next actions. The next action $a_{t+1}$is then chosen based on $MaxQ(s_{t+1}, a_{t+1})$. The model subsequently performs a gradient step by calculating the loss between the two terms in the Bellman Equation (10): $(1-\alpha).MaxQ(s_t, a_t)$ and $\alpha.[r_t(s_t, a_t)+\gamma.MaxQ(s_{t+1}, a_{t+1})]$, employing a loss function. Last but not least, an optimizer updates the weights of the model's neural networks based on the calculated loss. Finally, the target network is updated every n step.

Algorithm 1 DDQN

```
1:  Initialize the environment, network weights $\theta$, $\bar{\theta}$;
2:  Create experience replay memory;
3:  for episode = 1,2,..,  $N^{episodes}$ do
4:      Generate new Graph $G = (V, E)$ and initialize state $s_0$;
5:      while tour not ended or $T^t < 1$ do
6:          1- Select an action $a_t$ using $\epsilon$-greedy;
7:          2- Get reward $r_t$ and next state $s_{t+1}$;
8:          3- Store experience $< s_t, a_t, r_t, s_{t+1} >$ in experience memory
9:      end while
10:     if enough samples in experience replay then
11:         1- Sample a random batch of experiences;
12:         2- Compute $Q(s_t, a_t)$ and $Q(s_{t+1}, a_{t+1})$;
13:         3- Perform a gradient descent step $GradientLoss(Q(s_t, a_t), Q(s_{t+1}, a_{t+1}))$ using the sampled batch;
14:         4- Update DQN parameters $\theta$;
15:         5- Update target network parameters $\bar{\theta} = \theta$ every $N^{step}$ steps
16:     end if
17: end for
```

TABLE 3. Training parameters.

Parameter	Value
Batch size	64
Number of episodes	20000
Embedding dimension	32
Embedding iterations	1
Copy weights interval	100
Memory capacity	10000
n_step Q-Learning	2
Discount factor γ	0.9
Learning rate α	0.005
Learning rate decay rate	0.99998
Minimum epsilon	0.1
Epsilon decay rate	0.0006

D. MASKING SCHEME

During the DQN training, specifically the exploration phase, the model might return invalid actions in some states. For example, visit a CS to charge again even when there is no need to, just because it is the nearest node. In order to prevent the repetitive selection of invalid actions within full discrete action spaces, several works apply a technique known as masking scheme [24]. This technique implies to "mask out" infeasible actions and only let the model sample from the valid and feasible ones. Using this method, we can accelerate the training and effectively manage the vast space of actions. when masking, we should avoid explicitly instructing the model on how to behave in various states, which deviates from being model-free and learning from past experiences.

In this paper, we use this technique to hide infeasible solutions and avoid giving critical actions to the environment or the experience replay. The model applies the masking scheme in both exploration and exploitation. We suppose that an EV is at vertex v_t at step t, the model will compute all Q-values for all possible actions. These Q-values will then be passed to a masking function, which filters out invalid actions and returns the maximum Q-value among the valid ones that satisfy the following conditions.

- If a customer vertex $i \in Vc$ has already been visited (i.e., $d_i^t = 0$), it is considered redundant and masked.
- If the ending of time window e_i of a customer extends beyond the current system time $e_i \geq T^t$, it is not feasible to visit that customer, and thus it is masked.
- When the EV load can no longer accommodate the demand of any customer, all customers and charging stations ($Vc \wedge Vs$) are masked, leaving only the depot Vd as alternative.
- If the EV is currently at customer $i \in Vc$ and its battery SoC is insufficient to visit the next customer $j \in Vc$ and return to the depot, all customers (Vc) are masked. Only the CS (Vs) remain accessible.
- When the EV is at the depot, it is already fully charged. For this reason, all charging stations are masked to avoid selecting them as potential destinations. Conversely, if the EV is at a CS, the depot is masked to prevent it from being chosen as a destination. This condition is to avoid the back-and-forth trips between depot and CS. Even if the EV has no SoC to go to depot and needs to charge first, it still cannot visit a CS before depot. We noticed during the training the model adapts to this constraint.

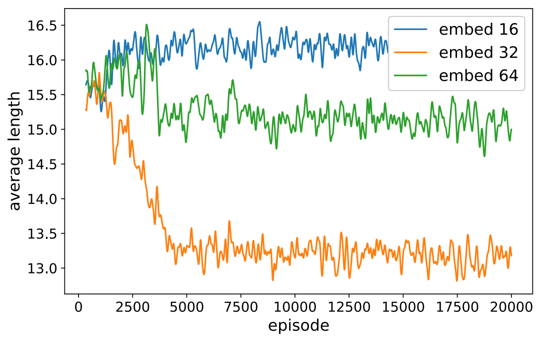
V. RESULTS

In this section, several case studies are carried out using both synthetic and real-world fleet traffic data to showcase the efficiency and effectiveness of the proposed model.

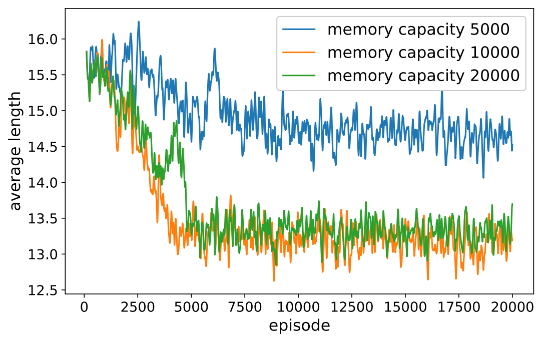
A. EXPERIMENTAL SETTING AND HYPER-PARAMETERS DISCUSSION

All the tests are performed on a desktop with i7-8700K CPU (3.70 GHZ and 64 RAM). The code is written in python and the DRL is coded in Pytorch 1.11.0. To evaluate the performance of the proposed model, we train it on various instances sizes ranging from n=10, 30, 50, to 100 customer nodes. The training is conducted using the set of hyper-parameters outlined in Table 3, which were identified as the optimal settings based on the comprehensive tests we have conducted. Additionally, we showcase the influence of hyper-parameters like embedding dimensions and memory capacity in Figure 3. Choosing an appropriate embedding dimension is crucial as it directly impacts the complexity and the expressiveness of the graph representation as illustrated in Figure 3a. If the embedding dimension is set too high, the graph representation becomes excessively intricate, potentially leading to increased computational complexity. On the other hand, if the embedding dimension is set too low, the graph representation may lack the necessary level of detail and fail to capture important characteristics of the graph data. Having a balanced embedding dimension ensures a graph representation that is both informative and able to generalize to unseen data. Based on our experiments, we have determined that a value of 32 for the embedding dimension provides a balanced representation that achieves good performance as illustrated in Figure 3a.

Moreover, we report the influence of the different memory capacity values in Figure 3b. The model relies on storing past experiences, and choosing an appropriate memory capacity is crucial. Increasing the memory capacity allows the agent to collect more representative samples, enabling it to learn from a wider range of experiences and potentially improve its overall performance. However, it is important to consider the computational abilities of the system when determining the memory capacity. Setting a memory capacity that is too large may result in reduced sampling efficiency, as the agent spends more time revisiting irrelevant or less informative experiences. On the other hand, setting the memory capacity too low leads to more frequent updates of stored experiences. While this ensures that the agent has access to recent information, it comes at the cost of losing valuable past experiences. This loss of historical information can hinder the agent's ability to generalize effectively and make informed decisions based on a broader context. Therefore, finding the optimal memory capacity involves striking a balance between storing enough past experiences to enable learning from a

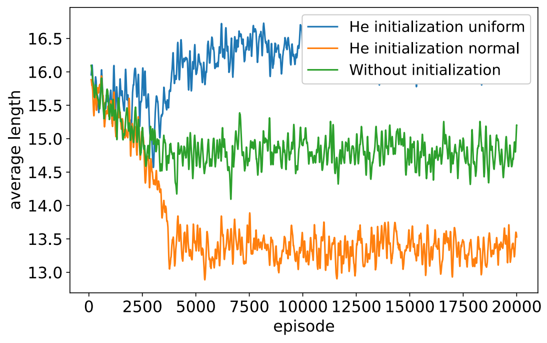


(a) Impact of the embedding dimension parameter on the model performance results

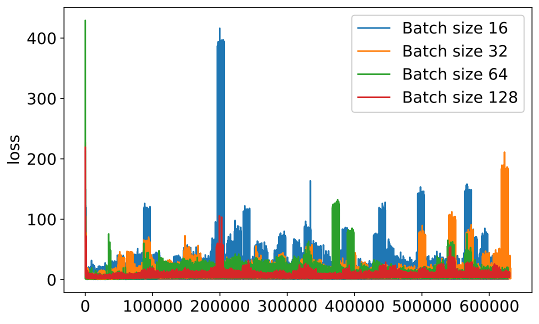


(b) Impact of the memory capacity parameter on the model performance results

FIGURE 3. Impact of the memory capacity and the embedding dimension parameters on the model performance results.



(a) Impact of initialization on the model performance results



(b) Impact of the batch size parameter on the model performance

FIGURE 4. Impact of the batch size and initialization on the model performance.

diverse samples and ensuring efficient computation. Based on our observations from the graph, we have determined that a large memory capacity does not significantly affect the training process. However, it is important to consider computational limits. For this reason, we have chosen a memory capacity of 10000 as it is slightly better during training and it requires less computational resources.

B. IMPACTS OF BATCH SIZE AND INITIALIZATION METHODS TO STABILIZE TRAINING

To conduct the training, random instances are generated similar to [13] to be able to compare our results. For each instance, we generate the vertices coordinates using a uniform distribution in a region [0,1] x [0,1]. The customers demands are selected randomly from {0.05, 0.10, 0.15, 0.20} with an equal probability. The system time, representing the planning horizon, is set within the range [0,1]. Regarding the time windows, we generate their centers uniformly in [0,1] and the length is normally generated with a mean of 0.2 and a standard deviation of 0.05. It is ensured that the time windows always fit within the planning horizon of [0,1]. Similarly, the test data is generated following the same methodology.

Furthermore, in order to stabilize the training, we chose to initialize all trainable variables, and for this, we conducted experiments with various initialization methods. Specifically, we tested Glorot and Bengio [25] and He et al. initialization [26], both with uniform and normal distributions. We also explored the option of not using any initialization. After evaluating the results, we found that the He-initialization with a normal distribution produced superior outcomes. The performance comparison is presented in Figure 4a. In addition, we performed tests using different loss calculation methods, including Mean Square Error (MSE) and Huber [27]. As a result, we decided to utilize Huber as it is better suited for minimizing distances [27] when following normal distribution, which was validated by our experiments.

It is worth noting that while some hyper-parameters may not directly influence training results, they can impact the stabilization of the training. For instance, the batch size can affect the fluctuations of the loss, as illustrated in Figure 4b. A larger batch size of 128 can lead to more stable training. By using a larger batch, the model updates its parameters based on a more representative samples, which reduces the impact of individual noisy or outlier samples, leading to more effective exploration and exploitation, however it is important to consider the computational abilities of the system. On the other hand, a smaller batch size of 16 can lead to more frequent updates, allowing the model to adapt quickly to recent experiences. Conversely, a larger batch size may provide a broader view of the environment, allowing

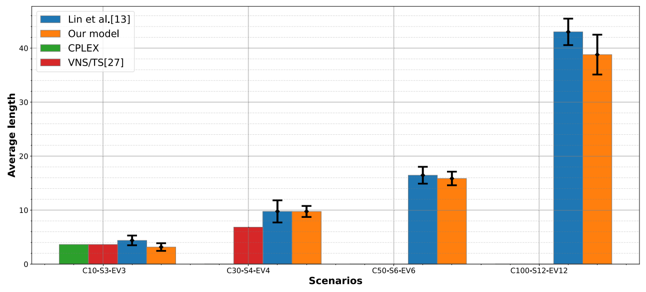


FIGURE 5. Comparison of average total distance.

TABLE 4. Comparison of average solution time in seconds.

Scenario	C10-S3 -EV3	C30-S4 -EV4	C50-S6 -EV6	C100-S12 -EV12
CPLEX	67.65	-	-	-
VNS/TS	10.37	536.80	-	-
RL (Lin et al.)	0.35	1.06	2.31	7.89
RL (Qiu et al.)	3.89	-	4.63	5.92
Our model	**0.03**	**0.12**	**0.23**	**1.05**

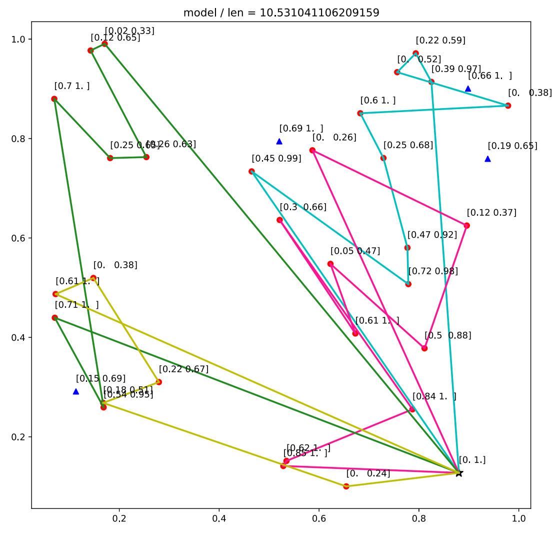


FIGURE 6. Model evaluation example with scenario C30-S4-EV4.

the model to generalize better and make more informed decisions across a wider range of situations. Based on our experiments, we have determined that a batch size of 64 stabilizes the training while maintaining efficient computational performance.

C. MODEL EVALUATION

In this section, we evaluate the performance of the proposed model by conducting several comparisons and analysing the results on both synthetic and real world data.

1) MODEL EVALUATION: SYNTHETIC DATA

The proposed model is evaluated and compared with [13] as both approaches address the same problem statement. Additionally, two conventional benchmarking strategies, VNS/TS [28] and CPLEX are also included in the comparison

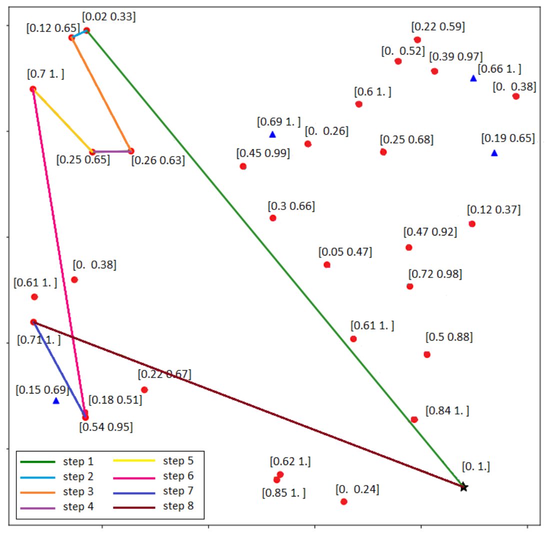


FIGURE 7. Example of the model's decision-making process.

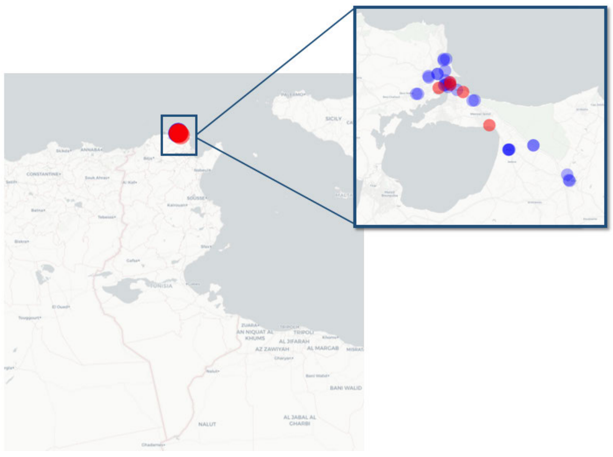


FIGURE 8. Section of the map of Bizerte with location of customers represented in blue, and charging stations represented in red.

in Figure 5, which results are reported by Lin et al. in [13]. We utilize the same three scenarios as in [13] to compare. For instance, the first scenario "C10-S3-EV3" with n=14 nodes denotes a graph with 1 depot, 10 customers, 3 charging stations and 3 EVs available in the fleet. For each scenario, we conduct 100 test instances, generated as explained in Section V-B, and we present the mean total distance in Figure 5 together with the gap between the average and the minimum value of total distance.

On small instances with 14 nodes, our model is able to provide better results than the greedy approach by Lin et al. achieving a lower total distance traveled. Although, when compared to RL models, heuristics still exhibit better solution quality despite being more time-consuming. However, the larger the number of nodes, the more heuristics struggle to provide feasible solutions. In contrast, both RL models generalize better and are able to report solutions. In this case, our proposed model outperforms the model developed by Lin et al. achieving a lower total traveled distance.

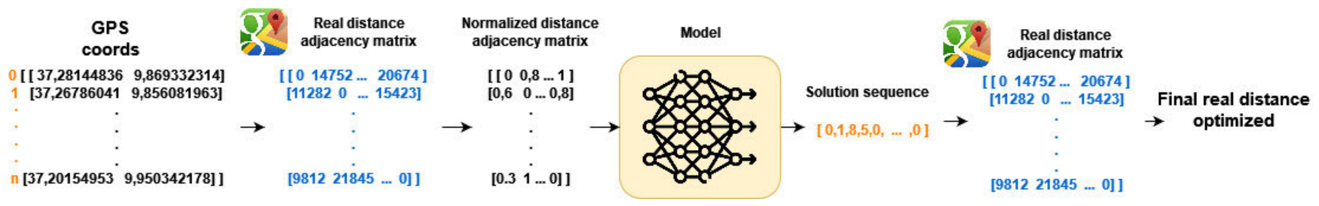
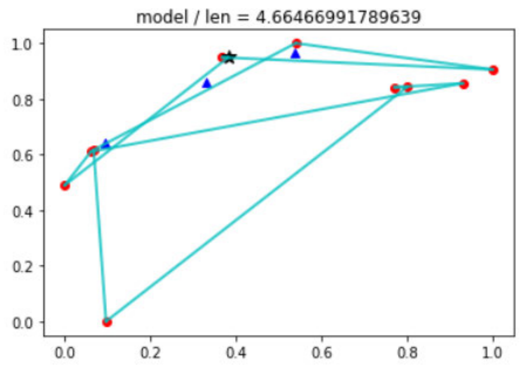


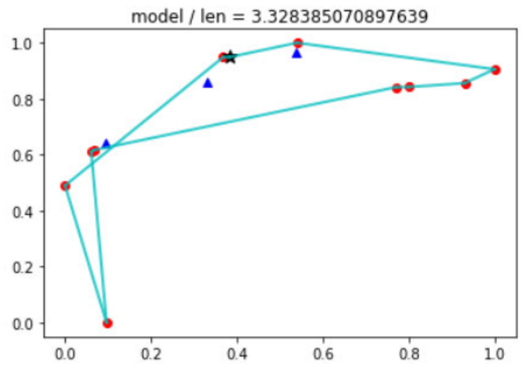
FIGURE 9. Procedure to calculate new real distances optimized by the model.

TABLE 5. Nodes visit order.

Blue EV		Pink EV		Green EV		Yellow EV	
Order	Node time window	Order	Node time window	Order	Node time window	Order	Node time window
1	[0.39 , 0.97]	1	[0 , 0.26]	1	[0.12 , 0.65]	1	[0 , 0.24]
2	[0.22 , 0.59]	2	[0.12 , 0.37]	2	[0.26 , 0.63]	2	[0.85 , 1.]
3	[0 , 0.52]	3	[0.5 , 0.88]	3	[0.23 , 0.65]	3	[0.18 , 0.51]
4	[0 , 0.38]	4	[0.5 , 0.47]	4	[0.7 , 0.1]	4	[0.22 , 0.67]
5	[0.6 , 1]	5	[0.61 , 1]	5	[0.54 , 0.95]	5	[0. , 0.38]
6	[0.25 , 0.68]	6	[0.3 , 0.66]	6	[0.71 , 1]	6	[0.61 , 1]
7	[0.47 , 0.92]	7	[0.84 , 1]				
8	[0.72 , 0.98]	8	[0.62 , 1]				
9	[0.45 , 0.99]	9	[0.85 , 1]				



(a) The original behaviour of the fleet



(b) The behaviour of the fleet when using our model

FIGURE 10. Analyzing the transformation in fleet behavior with the implementation of our model in scenario "C10-S3-EV3".

Another crucial aspect to consider is the runtime required for obtaining solutions. Table 4 showcases the average solution times in seconds across the 100 instances for various scenarios. To ensure fair comparisons, we employed comparable hardware configurations. Notably, our proposed model demonstrates superior performance compared to existing approaches. For instance, when solving the "C100-S12-EV12" scenario, the reinforcement learning model presented by Lin et al. [13] took 7.89 seconds, the model presented by Qiu et al. [14] took 5.92 seconds, whereas our approach achieved the same outcome in just 1.04 second. This substantial time reduction is attributed to the simplicity of our model architecture which achieves better results with fewer complexities. For instance, both RL models [13], [14] incorporate an Attention Mechanism, which inherently demands more processing time, despite the parallel computing used in the model proposed by Qiu et al. [14].

To illustrate an example of the proposed model behaviour, we report in figure 6 the evaluation of scenario "C30-S4-EV4". The visit order of nodes, along with their corresponding time windows (tw), is presented in Table 5. The model is able to visit all customers within their respected time windows using all the available EVs in the fleet. The number of EVs in solution is proposed and optimized by the model, which means less number of EVs will not be enough to respect customers demands and time window, and more will be excessive and leads to unnecessary energy consumption. Each colour represents an EV, the customers are represented in red dots, CS are in blue triangles and the depot is a black star.

In general, the model prioritizes nodes with approaching time windows, ensuring that visits are made within the specified time constraints. However, there are instances where the system time is significantly less than the end of the time windows, so the model prioritizes minimizing the traveled distance. We can observe an instance of the model's decision-making process and the sequence of actions it took by examining the green EV in Figure 7. During step 1, the model's attention was directed toward the farthest node

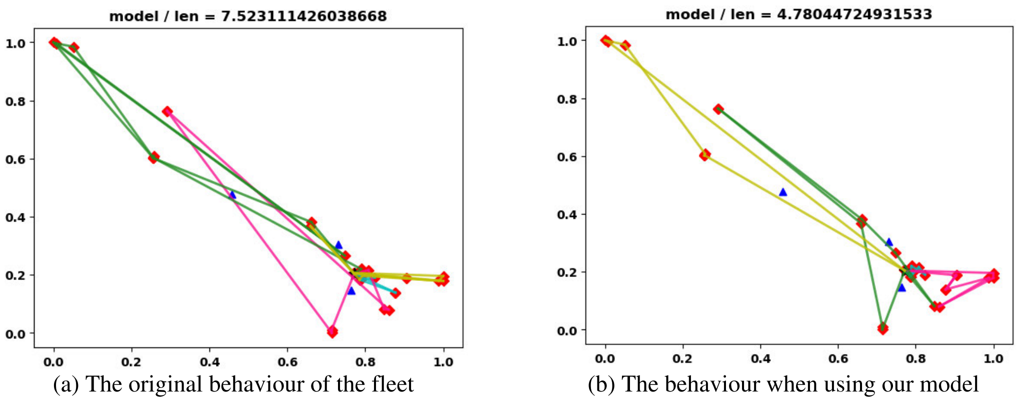


FIGURE 11. **Analyzing the transformation in fleet behavior with the implementation of our model in scenario "C30-S4-EV4".**

with the most urgent time window, which was set to close at 0.33. Subsequently, in step 2, the model faced a choice among three nearby nodes, each with time windows closing at approximately the same times: 0.65, 0.63, and 0.65. In this scenario, the model gave priority to the nearest node. Another example of the model behavior in the case of the blue EV. Starting from node 4, our model prioritizes visiting node 5, despite its time window ending at 1, because it is closer in proximity (can be observed in Figure 6). Next, the model visits node 6 with time window ending at 0.68 as there is still sufficient system time available. This trade-off allows the model to balance the urgency of time windows with the overall efficiency of the routing solution.

2) MODEL EVALUATION: REAL WORLD DATA

To evaluate the performance of our proposed model in real world settings, we utilized real-world data obtained from a public utility fleet company situated in Bizerte, Tunisia. To calculate the actual distance traveled between the company's different customers' locations, we followed a step-by-step process as illustrated in Figure 9. Firstly, we extracted real-world nodes coordinates from the company's GPS data and computed the distances between these nodes utilizing the Google Maps API. Secondly, we normalized these distances using the Min-Max Scaler method. The resultant normalized adjacency distance matrix served as input for our model to generate a solution. Lastly, leveraging the solution generated by the model, we recomputed the real distances values between nodes using Google Maps API.

In this section, we implemented two scenarios, "C10-S3-EV3" and "C30-S4-EV4," using the company's original GPS data collected on March 16th and 17th, 2023, respectively. Figure 8 provides a visual representation of the "C30-S4-EV4" scenario, where blue circles denote customer locations, and red circles mark the charging stations. First, in Figure 10 we compare the fleet's behavior and outcomes before and after applying our model to scenario "C10-S3-EV3." Initially, the fleet covered a total distance of 36.427 kilometers. However, following our model's optimization, this distance was reduced to 25.112 kilometers, representing a significant reduction of more than 31%. Second, in Figure 11 we analyze the outcomes for scenario "C30-S4-EV4". Initially, the fleet traveled a total distance of 205.147 kilometers. With the aid of our model, this distance was reduced to 106.897 kilometers, representing a remarkable gain of over 50% in distance efficiency.

VI. CONCLUSION

The EVRPTW is a crucial challenge that needs to be addressed for the successful adoption of EVs in fleets. In this paper we developed an RL framework capable of solving EVRPTW. RL algorithms, such as DDQN, offer a promising solution to the EVRPTW problem by allowing the EV fleet to learn optimal routing policies through trial and error. Additionally, incorporating graph-based modeling enhances navigation and optimization techniques, leading to more accurate and realistic modeling of the EVRP. The model is not only able to reduce the total traveled distance, but also to respect constraints like the capacity and the limited number of available vehicles, all while meeting the time window of the different customers. The model, characterized by streamlined complexity, is compared to state of the art solutions and exhibits better results in terms of time efficiency and minimized traveled distance. Although the results for small instances may not be optimal when compared to heuristics, the model is very promising in real world application particularly for large-size fleets where existing methods may not suffice. For instance, several components of the graph like demands, time windows or charging services availability, can change instantly. In this case, by making small adjustments, RL models have an efficient problem-solving ability that enables EV operators to quickly adapt and address the challenges posed by the stochastic nature of the EVPRTW.

However, there are several areas where further improvements can be made to enhance the model's performances. Utilizing real-world data for training could make it more applicable to real-world scenarios. Also, incorporating energy consumption considerations into the model and suggesting charging stations that utilize renewable energies represents a promising direction for future research. By continuously upgrading and refining the model, we can further enhance its effectiveness and contribute to the sustainable adoption of electric vehicles in fleet operations.

REFERENCES

[1] R. Baldacci, A. Mingozzi, and R. Roberti, ''Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints,'' *Eur. J. Oper. Res.*, vol. 218, no. 1, pp. 1–6, Apr. 2012.

[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018.

[3] A. I. Aygun and S. Kamalasadan, ''An optimal approach to manage electric vehicle fleets routing,'' in *Proc. IEEE Int. Conf. Power Electron., Smart Grid, Renew. Energy (PESGRE)*, Jan. 2022, pp. 1–6.

[4] R. Basso, B. Kulcsár, and I. Sanchez-Diaz, ''Electric vehicle routing problem with machine learning for energy prediction,'' *Transp. Res. B, Methodol.*, vol. 145, pp. 24–55, Mar. 2021.

[5] M. Dorokhova, C. Ballif, and N. Wyrsch, ''Routing of electric vehicles with intermediary charging stations: A reinforcement learning approach,'' *Frontiers Big Data*, vol. 4, p. 586481, May 2021, doi: 10.3389/fdata.2021.586481.

[6] A. L. C. Ottoni, E. G. Nepomuceno, M. S. D. Oliveira, and D. C. R. D. Oliveira, ''Reinforcement learning for the traveling salesman problem with refueling,'' *Complex Intell. Syst.*, vol. 8, no. 3, pp. 2001–2015, Jun. 2022.

[7] Y. Zhang, M. Li, Y. Chen, Y.-Y. Chiang, and Y. Hua, ''A constraint-based routing and charging methodology for battery electric vehicles with deep reinforcement learning,'' *IEEE Trans. Smart Grid*, vol. 14, no. 3, pp. 2446–2459, May 2023.

[8] T. M. Aljohani, A. Ebrahim, and O. Mohammed, ''Real-time metadata-driven routing optimization for electric vehicle energy consumption minimization using deep reinforcement learning and Markov chain model,'' *Electr. Power Syst. Res.*, vol. 192, Mar. 2021, Art. no. 106962.

[9] T. M. Aljohani and O. Mohammed, ''A real-time energy consumption minimization framework for electric vehicles routing optimization based on SARSA reinforcement learning,'' *Vehicles*, vol. 4, no. 4, pp. 1176–1194, Oct. 2022.

[10] Q. Xing, Y. Xu, and Z. Chen, ''A bilevel graph reinforcement learning method for electric vehicle fleet charging guidance,'' *IEEE Trans. Smart Grid*, vol. 14, no. 4, pp. 3309–3312, Jul. 2023.

[11] B. Turan, R. Pedarsani, and M. Alizadeh, ''Dynamic pricing and fleet management for electric autonomous mobility on demand systems,'' *Transp. Res. C, Emerg. Technol.*, vol. 121, Dec. 2020, Art. no. 102829.

[12] R. Basso, B. Kulcsár, I. Sanchez-Diaz, and X. Qu, ''Dynamic stochastic electric vehicle routing with safe reinforcement learning,'' *Transp. Res. E, Logistics Transp. Rev.*, vol. 157, Jan. 2022, Art. no. 102496.

[13] B. Lin, B. Ghaddar, and J. Nathwani, ''Deep reinforcement learning for the electric vehicle routing problem with time windows,'' *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 8, pp. 11528–11538, Aug. 2022.

[14] H. Qiu, S. Wang, Y. Yin, D. Wang, and Y. Wang, ''A deep reinforcement learning-based approach for the home delivery and installation routing problem,'' *Int. J. Prod. Econ.*, vol. 244, Feb. 2022, Art. no. 108362.

[15] M. Tang, B. Li, H. Liu, W. Zhuang, Z. Li, and J. Peng, ''Energy-oriented routing strategy of electric vehicle: An end-to-end reinforcement learning approach,'' in *Proc. 6th CAA Int. Conf. Veh. Control Intell. (CVCI)*, Oct. 2022, pp. 1–7.

[16] J. Jin and Y. Xu, ''Shortest-Path-Based deep reinforcement learning for EV charging routing under stochastic traffic condition and electricity prices,'' *IEEE Internet Things J.*, vol. 9, no. 22, pp. 22571–22581, Nov. 2022.

[17] A. Bogyrbayeva, S. Jang, A. Shah, Y. J. Jang, and C. Kwon, ''A reinforcement learning approach for rebalancing electric vehicle sharing systems,'' *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 7, pp. 8704–8714, Jul. 2022.

[18] M. Nazari, A. Oroojlooy, M. Takác, and L. V. Snyder, ''Reinforcement learning for solving the vehicle routing problem,'' in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst. (NIPS)*. Red Hook, NY, USA: Curran, 2018, pp. 9861–9871. [Online]. Available: https://dl.acm.org/doi/10.5555/3327546.3327651

[19] W. Kool, H. van Hoof, and M. Welling, ''Attention, learn to solve routing problems!'' in *Proc. ICLR Conf. Blind Submission*, 2019.

[20] H. Dai, B. Dai, and Le Song, ''Discriminative embeddings of latent variable models for structured data,'' in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 2702–2711.

[21] M. Xu, ''Understanding graph embedding methods and their applications,'' *SIAM Rev.*, vol. 63, no. 4, pp. 825–853, Jan. 2021.

[22] A. Bdeir, S. Boeder, T. Dernedde, K. Tkachuk, J. K. Falkner, and L. Schmidt-Thieme, ''RP-DQN: An application of Q-learning to vehicle routing problems,'' in *Proc. German Conf. Artif. Intell.*, 2021, pp. 3–16.

[23] J. Zhao, M. Mao, X. Zhao, and J. Zou, ''A hybrid of deep reinforcement learning and local search for the vehicle routing problems,'' *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 11, pp. 7208–7218, Nov. 2021.

[24] S. Huang and S. Ontañón, ''A closer look at invalid action masking in policy gradient algorithms,'' in *Proc. IEEE Conf. Games (CoG)*, 2020. [Online]. Available: https://arxiv.org/abs/2006.14171

[25] X. Glorot and Y. Bengio, ''Understanding the difficulty of training deep feedforward neural networks,'' in *Proc. 13th Int. Conf. Artif. Intell. Statist.*, vol. 9. PMLR, 2010, pp. 249–256.

[26] K. He, X. Zhang, S. Ren, and J. Sun, ''Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification,'' in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1026–1034.

[27] P. J. Huber, ''Robust estimation of a location parameter,'' *Ann. Math. Statist.*, vol. 35, no. 1, pp. 73–101, Mar. 1964.

[28] M. Schneider, A. Stenger, and D. Goeke, ''The electric vehicle-routing problem with time windows and recharging stations,'' *Transp. Sci.*, vol. 48, no. 4, pp. 500–520, Nov. 2014.

DIMETH NOUICER received the B.S. degree in computer science and the M.S. degree in data science from ISTIC, University of Carthage, Borj Cédria, in 2020. She is currently pursuing the Ph.D. degree with the Department of Electrical Engineering, ENSIT, University of Tunis, Tunis, Tunisia. Her research interests include operations research, artificial intelligence, and their applications in transportation, energy, and urban issues.

IKBAL CHAMMAKHI MSADAA received the Ph.D. degree in computer science and networking from TELECOM ParisTech, EURECOM, Sophia-Antipolis, France, in 2010. She was a Research and Teaching Assistant with EURECOM, from 2007 to 2010. She is currently an Assistant Professor of computer science with ENSTAB, University of Carthage. She is also a member of the LaRINa Research Laboratory, University of Carthage, where she serves as a Coordinator for WP3 within the REMO Project. Her work has been published in prestigious journals, such as IEEE COMMUNICATIONS SURVEYS AND TUTORIALS and *Mobile Networks and Applications*. Her research has been supported by national and international funding bodies including ANR-RNRT, France; the European Commission–7th Framework Program, DAAD, BMZ, Germany; and the Tunisian Ministry of Higher Education and Scientific Research. Her current research interests include the IoT, AI, UAV-assisted networks, and QoS management for wireless networks.

KHALED GRAYAA received the Ph.D. and H.D.R. degrees in communication systems from ENIT, ENSEEIHT, Toulouse, France, in 2000 and 2007, respectively. He was the Director of the National Higher Engineering School of Tunis (ENSIT), from 2011 to 2015. He is currently a Professor of communication systems and advanced electronics with ENSTAB, University of Carthage, and the Director of the LaRINa Research Laboratory. He is also a Tunisian Coordinator with the Renewable Energy-based E-Mobility in Higher Education (REMO) Project, DAAD, THI University, Germany; and AUI University, Morocco. He has published 25 articles in refereed conference lectures and 15 journal articles. His research interests include smart grids, AI, the IoT, and wireless communication systems.