Literature Survey

Dynamic Vehicle Routing using Deep Reinforcement Learning

Krishnan Jothi Ramalingam
Computational Sciences in Engineering
Matrikel-Nr. 4997812

March 20, 2021

Contents

List of Figures

Chapter 1

Introduction

1.1 Dynamic Vehicle Routing Problem

Dynamic vehicle routing problems (DVRPs) are problems in which a set of geographically dispersed customers is visited by one or more travelers or vehicles. The problems are dynamic because information stochastically changes over the operating horizon and there exist opportunities to make decisions in response to the revealed information. Eventhough conventional MDP models the stochasticity and dynamism of the problem, they still lacks route-based intuition, therefore route-based MDPs are introduced by *Ulmer et al.* Route-based MDP merges the concept of route plans with the conventional MDP model, since route plans can be an indicator of the potential impact of future realizations. The process of either updating the current route plan or determining a new route plan is executed whenever there is a change of state.

1.2 Solution methods

Solution methods for DVRP usually include reoptimization (RO), policy function approximation (PFA), lookahead algorithm (LA), and value function approximation (VFA). Recently Deep Reinforcement Learning methods are being used to solve DVRP problems, since they provide better results in handling the dynamism involved and the re-routing decision time is also comparatively low.

1.3 Structure of the survey

In this literature survey, three research papers are presented, which uses Reinforcement Learning techniques to solve DVRP. For each research paper, in the beginning a specific set of information like title, model, method, performance etc is presented, which serves the purpose of easy comparison between them. Required formulas, set of notations used in the research paper and pseudo code is presented wherever necessary. Gentle introduction about complex topics involved is also provided paralelly. Also at the end of each paper, future works section is also given in order to provide a futuristic view.

Chapter 2

DRL Approach to Solve DVRP with Stochastic Customers

Title: Deep Reinforcement Learning Approach to Solve Dynamic Vehicle Routing Problem with Stochastic Customers

Authors: Waldy Joe and Hoong Chuin Lau

Year: 2020

This approach(DRLSA) optimizes the **re-routing decision**.

Use case: Dynamic multi-vehicle same-day delivery routing problem with time windows and both known and stochastic customers.

Model: Route-based Markov Decision Process, with a state representation based on total cost of the remaining routes of the vehicles and the current time at the point of decision-making.

Method: DRLSA combines Deep Reinforcement Learning and Simulated Annealing based routing heuristic.

Performance comparison: DRLSA produces optimized re-routing decision almost instantaneously, based on small training episodes in the magnitude of thousands. When more than 50% of the total requests are dynamic, DRLSA outperforms Approximate Value Iteration(AVI) and Multiple Scenario Approach(MSC).

Note: *For simplification, it is assumed that a new request can be served without considering the package pickup. However, the author states that it is fairly straightforward to incorporate pickup in the problem setup.*

2.1 Overview

DRLSA combines neural-network based on-policy Temporal-Difference learning with experience replay to approximate the value function and a routing heuristic based on Simulated Annealing. DRLSA comprises of two phases, namely training phase and run-time. Before diving deep into the details, let us introduce few required terminologies.

The value function equation, $V(S_k)$ can be approximated to the following Bellman Equation,

$$\begin{gathered}\hat{V}(S_k) = min_{x \in X(S_k)}\{R(S_k, x) + \gamma \hat{V}(S_k^x, \theta)\}, \\ R(S_k, x) = Cost(S_k, x_k) - Cost(S_{k-1}, x_{k-1}), \\ Cost(S_k, x) = \sum_{m=1}^{M} \tau(\beta_m^x(k) + wait(\beta_m^x(k)) + pen(\beta_m^x(k))),\end{gathered}$$

where $R(S_k, x_k)$ is the reward function, $Cost(S_k, x)$ is the total cost (travel + wait + time window violation), and the description about other notations are provided in the table below.

The proposed state representation used in the route-based Markov Decision Process contains certain handcrafted features, which is shown below,

$$S_k'^x = \langle t(k), Cost_{remain}(S_k, x), Cost_{penalty}(S_k, x) \rangle,$$

where, $Cost_{remain}(S_k, x) = (Cost_{remain,m}(S_k, x))_{m \in M}$, $(Cost_{remain,m}(S_k, x)) = \tau(\delta_m^x(k) + wait(\delta_m^x(k))$, $Cost_{penalty}(S_k, x) = (Cost_{penalty,m}(S_k, x))_{m \in M}$, $(Cost_{remain,m}(S_k, x)) = pen(\delta_m^x(k))$

2.2 Training phase

The training phase is completely offline with simulated dynamic events in order to replicate the reality. Neural networks-based on-policy TD learning with experince replay is used to approximate the value function. The output of the training phase is a trained neural entwork or a non-parametric function, which approximates the value function for each post-decision state, $\hat{V}(S_k^x, \theta)$.

2.3 Run-time

During run-time, given the state S_k, the goal is to select the action/decision x_k^*, that returns the minimum $\hat{V}(S_k)$ (i.e. $x_k^* = argmin_{x \in X(S_k)}\{R(S_k, x) + \hat{V}(S_k^x, \theta)\}$). Since computing 'true' *argmin* involves brute force enumerating of all possible re-routing sequences, Simulated Annealing is used.

Simulated Annealing is a probabilistic technique for approximating the global optimum of a given function. Specifically, it is a metaheuristic to approximate global optimization in a large search space for an optimization problem. It is often used when the search space is discrete. It is a routing heuristic, which explores the search space via 2-opt local search probabilistically. Pseudo code of DRLSA algorithm is given below.

2.4 Future Works

Evaluating the algorithm on other large-sized real-world dynamic optimisation problems with different datasets.

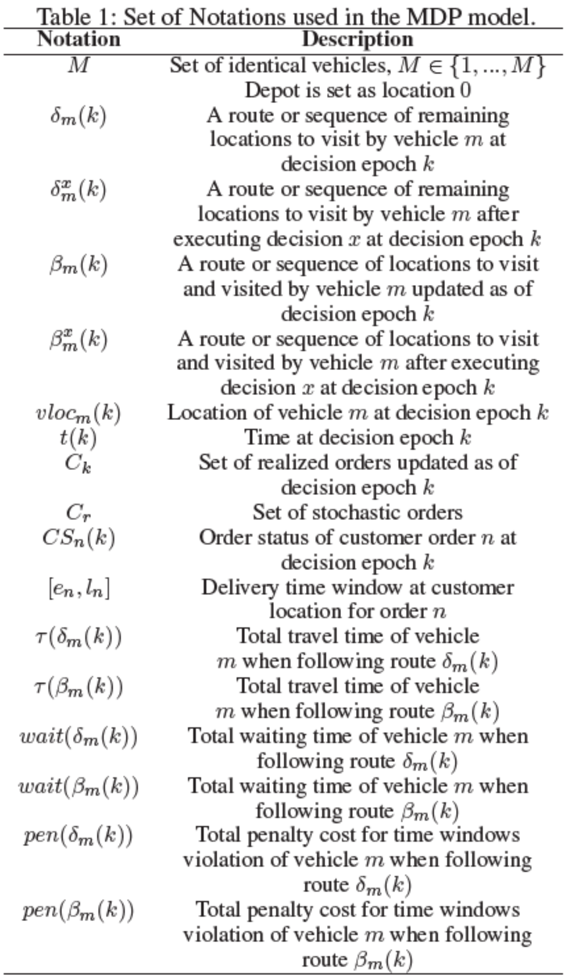
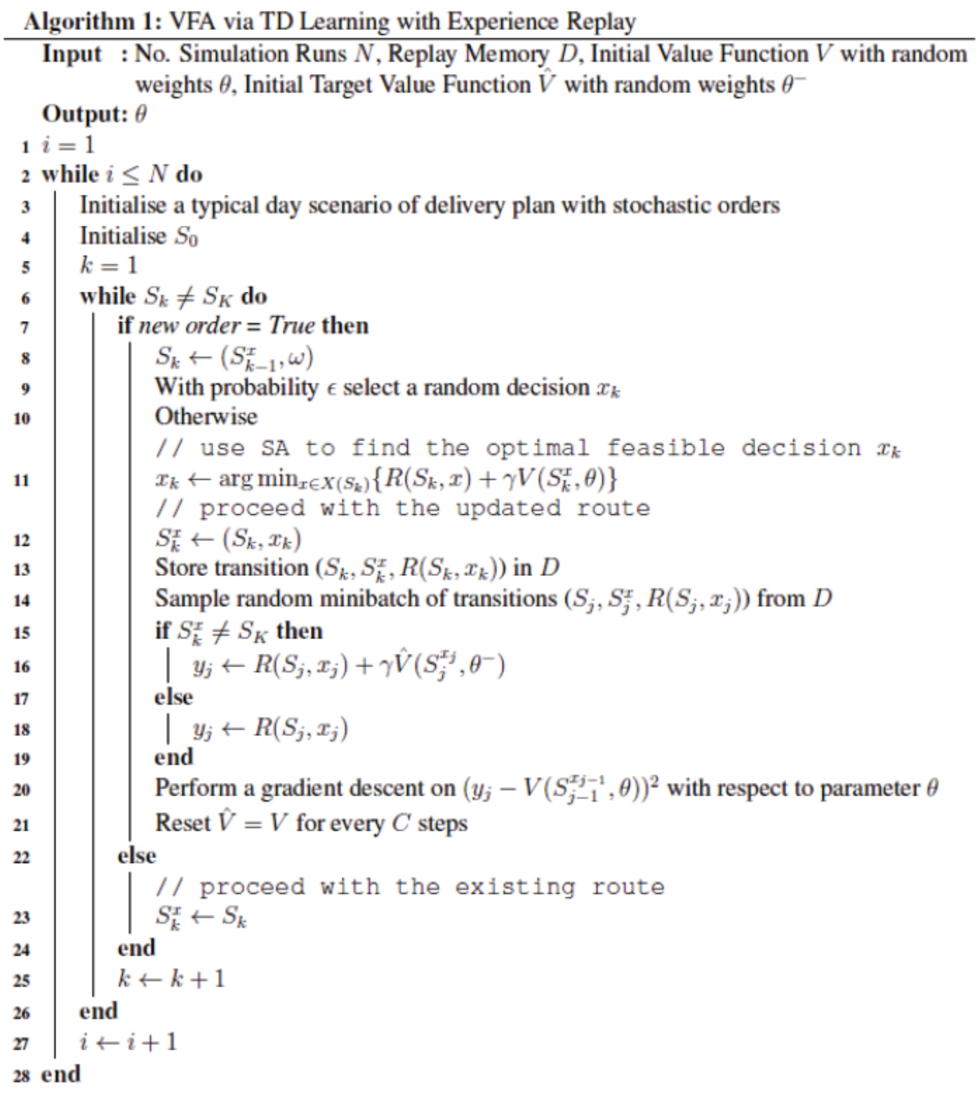


Table 1: Set of Notations used in the MDP model.

Notation	Description
M	Set of identical vehicles, $M \in \{1, ..., M\}$ Depot is set as location 0
$\delta_m(k)$	A route or sequence of remaining locations to visit by vehicle m at decision epoch k
$\delta_m^x(k)$	A route or sequence of remaining locations to visit by vehicle m after executing decision x at decision epoch k
$\beta_m(k)$	A route or sequence of locations to visit and visited by vehicle m updated as of decision epoch k
$\beta_m^x(k)$	A route or sequence of locations to visit and visited by vehicle m after executing decision x at decision epoch k
$vloc_m(k)$	Location of vehicle m at decision epoch k
$t(k)$	Time at decision epoch k
C_k	Set of realized orders updated as of decision epoch k
C_r	Set of stochastic orders
$CS_n(k)$	Order status of customer order n at decision epoch k
$[e_n, l_n]$	Delivery time window at customer location for order n
$\tau(\delta_m(k))$	Total travel time of vehicle m when following route $\delta_m(k)$
$\tau(\beta_m(k))$	Total travel time of vehicle m when following route $\beta_m(k)$
$wait(\delta_m(k))$	Total waiting time of vehicle m when following route $\delta_m(k)$
$wait(\beta_m(k))$	Total waiting time of vehicle m when following route $\beta_m(k)$
$pen(\delta_m(k))$	Total penalty cost for time windows violation of vehicle m when following route $\delta_m(k)$
$pen(\beta_m(k))$	Total penalty cost for time windows violation of vehicle m when following route $\beta_m(k)$

Figure 2.1: Notations and descriptions



```
Algorithm 1: VFA via TD Learning with Experience Replay
Input  : No. Simulation Runs $N$, Replay Memory $D$, Initial Value Function $V$ with random
         weights $\theta$, Initial Target Value Function $\hat{V}$ with random weights $\theta^-$
Output: $\theta$
$i = 1$
while $i \leq N$ do
    Initialise a typical day scenario of delivery plan with stochastic orders
    Initialise $S_0$
    $k = 1$
    while $S_k \neq S_K$ do
        if new order = True then
            $S_k \leftarrow (S^x_{k-1}, \omega)$
            With probability $\epsilon$ select a random decision $x_k$
            Otherwise
            // use SA to find the optimal feasible decision $x_k$
            $x_k \leftarrow \arg\min_{x \in X(S_k)} \{R(S_k, x) + \gamma V(S^x_k, \theta)\}$
            // proceed with the updated route
            $S^x_k \leftarrow (S_k, x_k)$
            Store transition $(S_k, S^x_k, R(S_k, x_k))$ in $D$
            Sample random minibatch of transitions $(S_j, S^x_j, R(S_j, x_j))$ from $D$
            if $S^x_k \neq S_K$ then
                $y_j \leftarrow R(S_j, x_j) + \gamma \hat{V}(S^{x_j}_j, \theta^-)$
            else
                $y_j \leftarrow R(S_j, x_j)$
            end
            Perform a gradient descent on $(y_j - V(S^{x_{j-1}}_{j-1}, \theta))^2$ with respect to parameter $\theta$
            Reset $\hat{V} = V$ for every $C$ steps
        else
            // proceed with the existing route
            $S^x_k \leftarrow S_k$
        end
        $k \leftarrow k + 1$
    end
    $i \leftarrow i + 1$
end
```

Figure 2.2: Pseudocode for DRLSA Algorithm

Chapter 3

Online Vehicle Routing With Neural Combinatorial Optimization and DRL

Title: Online Vehicle Routing With Neural Combinatorial Optimization and Deep Reinforcement Learning

Authors: James J. Q. Yu , Wen Yu and Jiatao Gu

Year: 2019

This approach optimizes both **service-offering** and **re-routing decision**.

Use case: Electric-driven autonomous vehicles as logistic carriers for VRP addressing dynamic systems, rendering conventional repetitive problem-solving process obsolete.

***Note**: This work is not limited to green logistic systems, other vehicles can be easily adopted with minor changes in the energy related definitions. This method can be adopted to all time-sensitive VRP and other constrained combinatorial optimization problems.*

Model: Dynamic system state S includes available requests, renewable generation outputs, the next stop of other vehicles in the system, and their battery charging demands if available. Complete tour (sequence of locations π to be visited) of a vehicle generated for a given dynamic system state S at a time instance t is the selected decision. Information regarding state transition is not provided.

Method: Applying Deep Reinforcement Learning to train the model parameters of a pointer network, which outputs the vehicle tours.

***Note**: During offline training, a multisampling scheme is devised to improve performance.*

Performance comparison: Proposed method can outperform conventional strategies in both static and dynamic logistic systems.
***Note**: Both pickup and delivery for a request is considered.*

3.1 Overview

A structural graph embedded pointer network is proposed to develop vehicle tours iteratively using deep reinforcement learning-based neural combinatorial optimization strategy. This approach comprises of 2 phases, offline training and online routing. Before diving deep into the details, let us introduce few required terminologies.

3.2 Tour graph

Each autonomous vehicle obtaines current dynamic system state at the beginning of each time slot from the Information center. Each vehicle can stop at any of the nodes in *stops*, which include pickup/delivery locations, all charging facilities and destination. Construction of tour graph for a vehicle k, requires enumeration of all possible tours among these *stops*.

***Note:** The nodes in the stops are encoded in such a way, that it include all necessary information of vehicles, logistic requests, and charging facilities.*

3.3 Pointer Network(PTRNET)

It comprises of two sub-networks, namely encoder and decoder.

3.3.1 Encoder

The input to the encoder includes the dynamic system state, system information and developed tour graph of each vehicle. STRUCT2VEC extracts the feature embeddings of nodes (handcrafted encodings) recursively according to the corresponding graph structure. The node embeddings are input into a recurrent neural network, which consist of LSTM cells. Finally, encoder produces an encode of the graph structure and node features.

***Note:** RNN is a class of Artificial Neural Networks which exhibit temporal dynamic behaviour by using their memory to process variabe length* *sequence of inputs. Embedding of the current location is input into RNN first, followed by the embeddings of other stops as input by a random order.*

3.3.2 Decoder

RNN of the decoder consists of LSTM cells. The input to the decoder is the encode generated by the encoder. Finally, the decoder constructs a complete tour iteratively.

Note: *Practically, RNNs are unable to handle "long-term dependencies", due to the problems of vanishing and exploding gradients. LSTMs are a special kind of RNN, which overcomes the above mentioned problems, by extending the memory of RNNs. LSTM has feedback connections, and in order to incorporate the long-term memory input, output and forget gates are used in the model architecture*

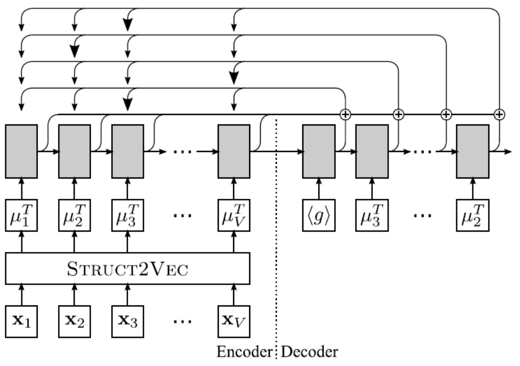


Figure 3.1: PTRNET and STRUCT2VEC based deep neural network

3.4 Offline training

Supervised learning requires enormous training data and extensive computation. Therefore DRL is employed, since it does not rely on optimal solutions.

Reward function of the online routing problem:

$$J(\Upsilon|S) = \mathbb{E}_{\pi \sim p_\Upsilon(.|S)}[O(\pi|S) - P(\pi|S)]$$

where,
Υ,π is the collection of all model parameters and tour respectively
$p_\Upsilon(.|S)$ is the probability distribution of $p(\pi|S)$ parameterized by Υ but without a deterministic π
. $O(\pi|S)$ is the objective reward, which gets increased for successful request delivery and decreased while traversing the network.
$P(\pi|S)$ is the constraint penalty function, which penalizes the total reward, if request are delivered late, on-board capacity exceeds the limit, on-board pending requests exists when arriving at the final location, the battery is

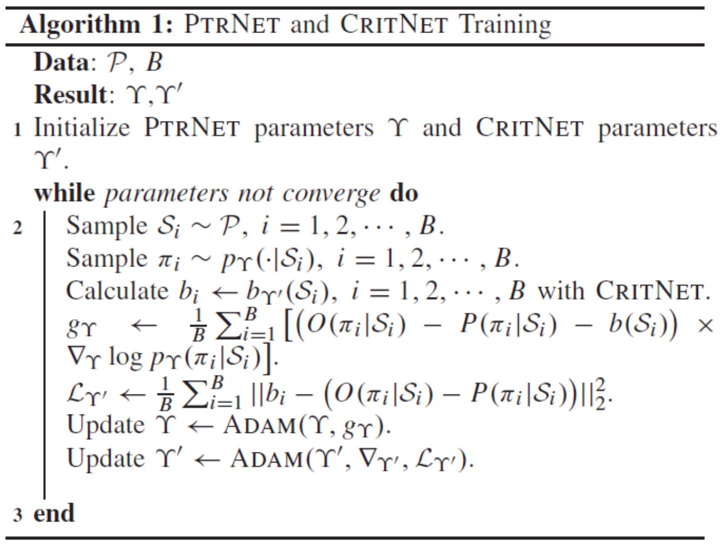
depleted or over-charged, the energy usage limit is exceeded at the renewable generation. More details about the reward function can be found in the literature.

Policy Gradient: The gradient of the reward function is formulated using REINFORCE algorithm. The gradient function is approximated with Monte Carlo sampling,

$$\nabla_\Upsilon J(\Upsilon) \approx \frac{1}{B}\sum_{i=1}^{B}[(O(\pi_i|S_i) - P(\pi_i|S_i) - b(S_i)) * \nabla_\Upsilon \log p_\Upsilon(\pi_i|S_i)]$$

where, B is the number of i.i.d samples $S_1, S_2, ..., S_B \sim P$, $\pi_i \sim p_\Upsilon(.|S_i)$, $b(S)$ is the baseline function which determines the expected reward function on system state S independent of π.

CRITNET is an auxiliary neural network to learn the expected reward values for given S, which uses the encoder network of the PTRNET. Υ' represent the model parameters of CRITNET. The output of this network is a scalare value $b(S)$ given S. Pseudo code is also provided paralelly.



Algorithm 1: PTRNET and CRITNET Training

```
  Data: $\mathcal{P}$, $B$
  Result: $\Upsilon$,$\Upsilon'$
1 Initialize PTRNET parameters $\Upsilon$ and CRITNET parameters $\Upsilon'$.
  while parameters not converge do
2     Sample $\mathcal{S}_i \sim \mathcal{P},\ i = 1, 2, \cdots, B$.
      Sample $\pi_i \sim p_\Upsilon(\cdot|\mathcal{S}_i),\ i = 1, 2, \cdots, B$.
      Calculate $b_i \leftarrow b_{\Upsilon'}(\mathcal{S}_i),\ i = 1, 2, \cdots, B$ with CRITNET.
      $g_\Upsilon \leftarrow \frac{1}{B}\sum_{i=1}^{B}\big[\big(O(\pi_i|\mathcal{S}_i) - P(\pi_i|\mathcal{S}_i) - b(\mathcal{S}_i)\big) \times \nabla_\Upsilon \log p_\Upsilon(\pi_i|\mathcal{S}_i)\big]$.
      $\mathcal{L}_{\Upsilon'} \leftarrow \frac{1}{B}\sum_{i=1}^{B} ||b_i - \big(O(\pi_i|\mathcal{S}_i) - P(\pi_i|\mathcal{S}_i)\big)||_2^2$.
      Update $\Upsilon \leftarrow$ ADAM$(\Upsilon, g_\Upsilon)$.
      Update $\Upsilon' \leftarrow$ ADAM$(\Upsilon', \nabla_{\Upsilon'}, \mathcal{L}_{\Upsilon'})$.
3 end
```

Figure 3.2: PTRNET and CRITNET Training

3.5 Online routing

Given system state S and trained PTRNET, the final tour π can be constructed efficiently by sampling M multiple candidate tours (*Multisampling*) and the one with the largest reward is retained.

3.6 Future Works

Deploying the algorithm to solve large-scale combinatorial optimization problems and employing other advanced Deep neural network or DRL techniques in the proposed strategy.

Chapter 4

RL Benchmarks for Online Stochastic Optimization

Title: ORL: Reinforcement Learning Benchmarks for Online Stochastic Optimization Problems

Authors: Bharathan Balaji, Jordan Bell-Masterson, Enes Bilgin, Andreas Damianou, Pablo Moreno Garcia, Arpit Jain, Runfei Luo, Alvaro Maggiar, Balakrishnan Narayanaswamy and Chun Ye

Year: 2019

This approach optimizes both **service-offering** and **re-routing decision**.

Use case: Stochastic and dynamic capacitated multi-vehicle routing problem with pickup and delivery, time windows and service guarantee.

Model: Formulated mathematical model is Markov Decision Process. Details about the reward function, action space and the features of hand-crafted state representation is explained in detail in the following text.
***Note**: No information regarding state transition is given.*

Method: Deploying Reinforcement Learning to train a policy using a two-layer neural network with 512 hidden units each.
***Note**: Simulation environment is from OpenAI Gym interface and integrated with the RLlib library.*

Performance comparison: Reinforcement learning approach outperforms the so called baseline algorithm, which is the modified version of three-index Mixed Integer Programming(MIP) formulation.
***Note**: Authors open sourced the code with the flexibility of comparison with other algorithms. Researchers can also replicate the results, test algorithms and fine-tune hyperparameters.*

4.1 Overview

APE-X DQN algorithm is used to train the policy due to its ability to scale by generating more experience replays and picking from them in a distributed prioritized fashion. A two-layer ANN with 512 hidden units each is used to reperesent the policy. APE-X DQN algorithm is designed for simulated environments, where it is possible to generate large quantitites of data in parallel.
Note: *Hyperparameters of the algorithm are directly taken from Atari APEX example provided in RL-lib.*

4.2 MDP Model

4.2.1 State

$$S_t = (p_t, h_t, c_t, l_t, w_t, e_t, v_t)$$

where, S_t is the state, p_t is the pickup location, h_t is the driver's position, c_t is the capacity left, l_t is the orders' location, w_t is the status of the request (open, accepted, pickedup or delivered/inactive), e_t is the time elapsed after generation of each order, v_t is the corresponding dollar value.

4.2.2 Action

Action A_t is chosen from five options $(a_1, a_2, a_3, a_4, a_5)$ for a given state S_t where,
a_1 - accept the open order $i \in P$
a_2 - pick up the accepted order $i \in A$
a_3 - deliver the in transit order $i \in T$
a_4 - the pickup location $j \in R$
a_5 - wait and stay unmoved

4.2.3 Reward

Reward R_t is the difference between the delivered orders f_t and the cost q_t.

$$R_t = \tfrac{1}{3}(\mathbb{1}_{accepted} + \mathbb{1}_{picked-up} + \mathbb{1}_{delivered})f_t - q_t$$

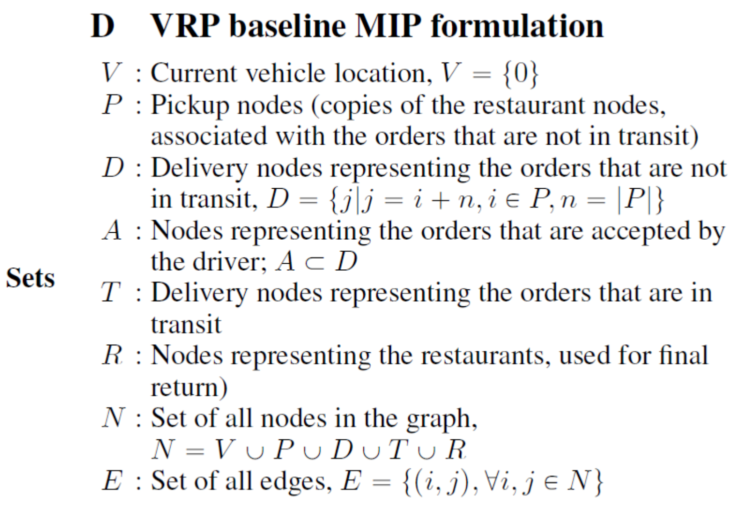
where f_t is divided into 3 parts for the purpose of reward shaping.

$$q_t = (q_{time} + q_{move} + q_{failure})$$

where $q_{time}, q_{move}, q_{failure}$ is the time cost, moving cost(per time step) and time window violation penalty respectively.
Note: *Invalid actions: cannot pick up an order if $c_t = 0$, unaccepted orders can't be picked up, unable to deliver an order that is not in transit.*

4.2.4 Methodology

The output of a policy is the action to be chosen, for a given input state. Multiple policies are trained by modifying the scale of the problem (map size) $\in \{5*5, 8*8\}$, maximum number of orders $\in \{5, 10\}$, number of pick-up locations $\in \{2, 3\}$, order generation probability for 4 different zones in the city $\in \{(0.1, 0.5, 0.3, 0.1), (0.5, 0.3, 0.1, 0.1)\}$. These modifications are made to test the generalizability of the algorithm. In all the cases, RL algorithm outperforms the baseline accross different instance sizes, and generalizes well for unseen order patterns.



D VRP baseline MIP formulation

Sets

V :	Current vehicle location, $V = \{0\}$
P :	Pickup nodes (copies of the restaurant nodes, associated with the orders that are not in transit)
D :	Delivery nodes representing the orders that are not in transit, $D = \{j \mid j = i + n, i \in P, n = \lvert P \rvert\}$
A :	Nodes representing the orders that are accepted by the driver; $A \subset D$
T :	Delivery nodes representing the orders that are in transit
R :	Nodes representing the restaurants, used for final return)
N :	Set of all nodes in the graph, $N = V \cup P \cup D \cup T \cup R$
E :	Set of all edges, $E = \{(i, j), \forall i, j \in N\}$

Figure 4.1: Notation and description

Note: *Details about the baseline algorithm is provided in the Appendix D of the paper.*

4.3 Future Works

Deploying this strategy to real-world industrial problems and employing various other reinforcement learning algorithms and deep neural network architectures.