Multi-Agent Learning of Efficient Fulfilment and Routing Strategies in E-Commerce

Kavin 2022111027
Shivani 2022101019
Vaishnavi 2022102070

Introduction

- With the rise of e-commerce, the efficient operation of supply chains has been facing new problems.
- Primary of those, being that the traditional supply chain network of warehouses and stores have been extended to include individual customer locations.
- The current problem can be broken down into two parts.
- The selection of a warehouse from which the order is to be served.
- The routing of vehicles from these nodes to a set of delivery locations.
- Both of these decisions are currently based on simple heuristics.

Overview

- We claim that both these decisions are interdependent and propose a RL algorithm to perform these tasks.
- The task of the RL algorithm is to pick the warehouse for fulfilling each order, or choose to defer the fulfillment to a future time.
- All orders being fulfilled at the current time step are then serviced by a vehicle routing agent, which takes into account constraints like vehicle capacity, travel times, and customer time windows

Assumptions

- There is only one type of product of which any amount up to the vehicles maximum capacity can be ordered by each customer.
- Each delivery vehicle has a uniform maximum capacity and travels at a constant speed.
- Each warehouse can dispatch an unlimited number of vehicles and each dispatched vehicle must return to the starting warehouse after finishing service
- Customers arrive at via a stochastic process in random locations with demands drawn from a uniform distribution. The total number of customers is also randomly sampled. Warehouses have fixed locations. Inventories are replenished periodically via an external process

Objective

- We want to minimise the number of vehicles and trips needed while satisfying the constraints.
- Hence we need to find total distance J that minimises

$$J = \min_{a_*, f_*, l_*} \left(\sum_{i,j,k} d_{i,j} a_{i,j,k} + \sum_{i,k} d_{o,i} f_{i,k} + \sum_{i,k} d_{o,i} l_{i,k} \right)$$

  where $d_{i,j}$ is the distance from customer c_i to c_j, $d_{o,i}$ is the distance from origin (depot) to c_i. $a_{i,j,k}$, $f_{i,k}$, and $l_{i,k}$ are indicator variables.

- If vehicle k serves c_i directly after c_j, $a_{i,j,k} = 1$. If c_i is the first customer to be served by vehicle k, $f_{i,k} = 1$. If c_i is the last customer to be served, $l_{i,k} = 1$.

Constraints and notation

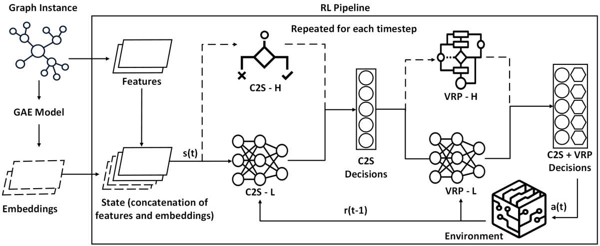
- Splitting a customers order between multiple deliveries is not allowed.
- Items must be delivered within a time window specified by the customer, i.e. The delivery time to customer c_i in vehicle k must satisfy $T_{i,min} \leq t_{i,k} \leq T_{i,max}$.
- The total load of vehicle k cannot exceed its maximum capacity Q.

$$\sum_{i,j} m_j a_{i,j,k} + \sum_{i} m_i f_{i,k} \leq Q \quad \forall k$$

- Each customer requires service time Δ for their delivery.
- v is the fixed vehicle speed. The above constraints imply

$$t_{i,k} \geq \frac{d_{o,i}}{v} \quad \text{if } f_{i,k} = 1 \text{ and} \quad t_{i,k} \geq t_{j,k} + \Delta + \frac{d_{j,i}}{v} \quad \text{if } a_{j,i,k} = 1$$

Methodology



- Our proposed solution consists of three parts.
- Training a Graph Auto Encoder (GAE) to obtain the node embeddings of the graph.
- Using these learnt embeddings along with the other state features to train a Deep Q-Network(DQN) agent to assign warehouses to customers. We'll refer to this agent as the *C2S* agent.
- We finally have an agent to compute the required routes and assign vehicles. We'll refer to this agent as the *VRP* agent.

What is a Graph Autoencoder (GAE)?

- We aim to find **embeddings** for each customer node to represent the relationships between customers and warehouses.
- The goal is to learn a **compact representation** of the graph to analyze customer proximity and connections.
- A GAE is designed to generate **low-dimensional embeddings** for nodes in a graph.
- The **encoder** takes node features and graph structure (edges) as input and generates embeddings.
- The **decoder** reconstructs the graph by predicting whether an edge exists between two nodes based on their embeddings.

Encoding in GAE

- The **input graph** consists of:
  - **Nodes** representing customer locations.
  - **Edges** representing proximity between customers based on warehouse range.

- The **encoder** takes this graph and clusters it into groups based on proximity. Then, we produce an **embedding** for each customer node.

The following is the clustering Algorithm:

Data: Customer locations, parameter $n \in \mathbb{Z}^+$
Result: Clusters, neighbourhood radius ρ
Compute Euclidean distances $d_{i,j}$ between customers;
Initialise set $\mathcal{K}$ of clusters as empty set;
while *at least one customer has no cluster mapping* **do**
  Define a new empty cluster Φ;
  Add nearest unmapped customer from depot to Φ;
  for *all customers in* Φ **do**
    Add nearest n neighbours to Φ if these neighbours have no existing mapping;
    if *no new customers got added to* Φ **then**
      break;
    end
  end
  Add cluster Φ to $\mathcal{K}$;
end
Set neighbourhood radius ρ as half of the largest cluster diameter in $\mathcal{K}$;
Finalise: Set of clusters $\mathcal{K}$ and radius ρ;

Algorithm 2: Cluster preprocessing pseudo-code.

Figure: Here: we take rho distance from each warehouse as the clusters

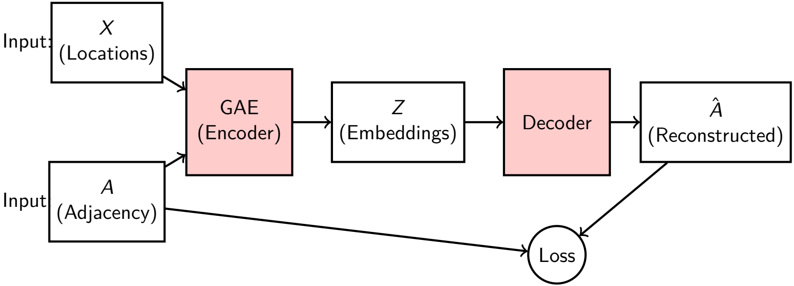
Decoding in GAE

- The **decoder** reconstructs the graph by calculating the **similarity** between node embeddings.
- Similarity is computed using **Euclidean distance**:

$$\text{similarity} = 1 - \frac{\|e_1 - e_2\|_2}{\|e_1 - e_2\|_{\max}}$$

- The output layer produces a **2-dimensional embedding** space for each node, representing customer locations in a compressed form.
- A high similarity indicates an edge between the nodes (i.e., customers are close in space).

Diagram



Graph Structure

- X: Feature matrix containing customer locations (Here x and y axis).
- A: Adjacency matrix representing customer connections after clustering.
- Z: 2-Dimensional Embeddings
- $\hat{A}$: Reconstructed adjacency matrix after decoding.

$C2S$ Learning Agent

- We model the $C2S$ problem as a MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$.
- The state $\mathcal{S}$ consists of features for each customer(GAE representation, Distance from warehouses, etc).
- The $\mathcal{A}$ is a mapping from a customer c_i to a warehouse j or deferment.
- $\mathcal{T}, \mathcal{R}, \gamma$ are the transition probabilities, set of rewards and discount factor respectively.
- Proceeding in a FCFS order, the RL agent computes a decision for each customer, including ones that have been previously deferred.
- After all the decisions have been, the information is passed to the *VRP* agent.

Reward for $C2S$ Learning Agent

- The reward given to the $C2S$ agent for a customer c_i can be split into a few parts.
- A negative reward D_i proportional to the straight line distance from the warehouse to the customer. The range of D_i is $[-2.12, 0]$.
- A negative reward $L_i \propto -\frac{Z}{r}$, where the vehicle assigned to c_i has a round-trip distance of Z for a trip that serves r customers. The range of L_i is $[-1, 0]$.
- A fixed fulfillment reward $F_i = 1$ when customer c_i is assigned to a warehouse, and 0 if the customer is deferred to a future time step.
- A negative reward U_i proportional to the empty space on the vehicle when it starts on a trip, given equally to all the customers served in that trip. It also falls in the range $[-1, 0]$.

Reward for $C2S$ Learning Agent

- The overall reward function is given by

$$reward(c_i) = a_1(D_i + L_i) + F_i + a_2 U_i$$

Here a_1 and a_2 are user defined constants.

- The reward function when we defer the fulfillment of an order is given by

$$reward(c_i) = \gamma^h(a_1(D_i + L_i) + F_i + a_2 U_i)$$

Here h denotes the number of times service for deferred for c_i.

- There is also a fixed penalty of -10 is assigned if a customer is dropped completely.

- Both L_i and U_i depend on the route computed by the *VRP* agent.

- The $C2S$ agent uses a Deep Q-Network to perform the actual mapping of customers to warehouses.

How DQNs Work for C2S? (Overview)

- **State Representation:**
  - A 19-dimensional vector is constructed for each customer: s = $\{z_1, z_2,\ d_1, \dots, d_4,$ inventory$_1, \dots,$ inventory$_4$, demand, $T_{\min},\ T_{\max},\ t,$
  - z_1, z_2: GAE embeddings of the customer.
  - d_i: Distance between the customer and each warehouse.
  - $T_{\min}, T_{\max}$: Time window for delivery.
  - t: Current environment time.
- **Action Space:**
  - $A = \{w_1, w_2, w_3, w_4, \text{defer}\}$:
    - w_i: Assign customer to warehouse i.
    - Defer: Postpone customer allocation to a future time step.
- **Neural Network Architecture:**
  - Input: 19-dimensional state vector s.
  - Layers:

$$\begin{aligned} h_1 &= \tanh(W_1 s + b_1) \quad \text{(First hidden layer, 76 units)} \\ h_2 &= \tanh(W_2 h_1 + b_2) \quad \text{(Second hidden layer, 38 units)} \\ Q(s, a) &= W_3 h_2 + b_3 \quad \text{(Output layer, 5 actions)} \end{aligned}$$

How DQNs Work for C2S? (Training Process)

- **Q-Learning Objective:**
  - Q-values are updated using the Bellman equation:

$$Q(s, a) \leftarrow r + \gamma \max_{a'} Q(s', a')$$

  - Mean Squared Error (MSE) loss:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \left(Q(s_i, a_i) - \left[r_i + \gamma \max_{a'} Q(s_i', a') \right] \right)^2$$

- **Training with Replay Buffer:**
  - Store experiences $(s, a, r, s', \text{done})$ in the replay buffer.
  - Sample a minibatch and compute loss.
  - Update network weights via backpropagation using the Adam optimizer.
- **Replay Buffer:**
  - Stores transitions $(s, a, r, s', \text{done})$.
  - Enables random sampling for uncorrelated training batches.

VRP Learning Agent Overview and Base Policy

- Train a neural network to get a base policy to be used in rollout
- Use rollout to choose a tour (a set of vehicle-customer pairs and a delivery path)
- Optimise this chosen tour using forward SAT
- After all customers are part of a tour, optimise all tours using tightening SAT
- Each customer is assigned to a cluster. (The number of clusters is not decided in advance.) This information is used to train the base policy.
- Let ρ be the largest cluster diameter. Let τ be the median time taken to travel between any two pairs of customers.

VRP Agent Roll-out Reward

- Let vehicle k serve customers $p \in 1, 2, \cdots P$. The distance of each leg in the route is d_p, and the time between customer services is t_p.

- The reward for each step in the route is:

$$R_{k,p} = \frac{\rho - d_p}{d_{max}} + \frac{\tau - t_p}{t_{max}} + \gamma^{P-p} R_{term}$$

  We penalise longer step distances and travel times. The discount factor γ is raised to power $P - p$ to account for remaining customers.

- The terminal reward is:

$$R_{term} = 2\rho - \frac{1}{P+1}\left(\sum_p d_p + D_{return}\right)$$

  The return distance D_{return} is the distance from the last customer to the depot. This reward is meant to penalise long routes.

Algorithm – 1 VRP Step

Data: Customer data specification
Result: Vehicle routes and service times
Initialise: Single vehicle at depot, parameter κ;
while *at least one customer yet to be served* **do**
  identify further feasible customer-vehicle pairs;
  shortlist top κ pairs identified by RL;
  do stochastic rollouts using RL policy;
  choose the decision with the lowest total distance;
  pick sub-tour being served by chosen vehicle;
  optimise sub-tour using forward SAT;
  if *vehicle leaving depot* **then**
    spawn a new vehicle at depot;
  end
  implement the optimised sub-tour;
end
Finalise: Optimise vehicle tours with tightening SAT;

Algorithm 1: Running an episode of CVRP-TW. The same procedure is used during training and testing, with the exception that exploration steps are taken in a uniformly random fashion instead of the learnt RL policy.

Figure: Algorithm 1

Algorithm – 2 VRP clustering

Data: Customer locations, parameter $n \in \mathbb{Z}^+$
Result: Clusters, neighbourhood radius ρ
Compute Euclidean distances $d_{i,j}$ between customers;
Initialise set $\mathcal{K}$ of clusters as empty set;
while *at least one customer has no cluster mapping* **do**
  Define a new empty cluster Φ;
  Add nearest unmapped customer from depot to Φ;
  for *all customers in* Φ **do**
    Add nearest n neighbours to Φ if these neighbours have no existing mapping;
    if *no new customers got added to* Φ **then**
      break;
    end
  end
  Add cluster Φ to $\mathcal{K}$;
end
Set neighbourhood radius ρ as half of the largest cluster diameter in $\mathcal{K}$;
Finalise: Set of clusters $\mathcal{K}$ and radius ρ;

Algorithm 2: Cluster preprocessing pseudo-code.

Figure: Algorithm 2

VRP Learning Agent MAX-SAT

- Once rollout is complete, the sub-tour is optimised using forward SAT. We order customers in the sub-tour R using order index variables O_i. If D_i denotes step lengths, we want to optimise

$$J = \min_{O_i} \left[\left(\sum_R D_i \right) + D_{return} \right]$$

With constraints

$$D_i = \begin{cases} d_{loc,i} & \text{if } O_i = 1 \\ d_{j,i} & \text{if } O_i < 1, O_i = O_j + 1 \\ d_{max} & \text{if } O_i = -1 \text{ i.e. customer not serverd} \end{cases}$$

$$(R_i - \Delta \leq O_i \leq R_i + \Delta) \text{ and } O_i \geq 1 \; \forall c_i \in R$$

- After all customers are part of a tour, we optimise all tours using tightening SAT.

Vehicle Routing Problem Heuristic (VRP-H): Overview

- **Purpose:**
  - VRP-H determines an efficient route for delivering customer orders.
  - Adheres to constraints like:
    - Vehicle capacity,
    - Customer time windows.
- **Key Steps:**
  1. Sort customers assigned to the warehouse by time window opening.
  2. If no vehicle is at the depot, spawn a new vehicle.
  3. Choose the first feasible customer from the sorted list based on:
     - Travel time,
     - Time window constraint,
     - Remaining vehicle capacity.

Vehicle Routing Problem Heuristic (VRP-H): Details

- **Key Steps (continued):**
  4. Serve the customer, update the time and vehicle availability.
  5. Repeat until no feasible customers are left.
- **Characteristics:**
  - Prioritizes simple heuristics over optimality.
  - Ensures practical constraints are met efficiently.
  - Complements the C2S-H policy by determining vehicle routes for assigned customers.

Baselines

- To evaluate the performance of the RL model, we define a few baseline heuristics to compare against.
- $C2S - H$: Allocates each customer to the nearest warehouse immediately upon their generation.
- $VRP - H$: Sorts the assigned customers according their time window opening. The choice of next served is based on travel time, time window constraint, and demand quantity.
- We then consider combinations of these two heuristics with our two agents, $C2S - L$ and $VRP - L$.

Combinations of settings for training

Table 1: Agent versions with description: values in superscript indicate reference sections.

Agent	Description
C2S-H + VRP-H	Combination of the C2S-Heuristic and VRP-Heuristic[5], serving as a comprehensive heuristic baseline
C2S-L + VRP-H	Utilizes the C2S Agent[4.2] along with the VRP-Heuristic[5]
C2S-H + VRP-L	Incorporates the C2S-Heuristic[5] and the VRP-Agent defined in[4.3]
C2S-L + VRP-L	Signifies the use of both agents as learning-based for C2S and VRP.
C2S-P + VRP-L	We use a pre-trained model from C2S-L + VRP-H to set the weights for the C2S agent, and the VRP agent also adopts a learning-based approach. Our training strategy unfolds in two phases: first, we exclusively train the C2S Agent with VRP-H, and subsequently, we train the VRP Agent based on decisions made by the pre-trained C2S Agent without further training.

Figure: Combinations of settings

Training Parameters and metrics

- C2S: Batch size 512, Leaning rate 0.001, Adam Optimizer, Epsilon decay 0.97, Gamma 0.9, Buffer capacity 10^5
- VRP: Batch size 512, Leaning rate 0.001, Adam Optimizer, Epsilon decay 0.999, Gamma 0.9, Buffer capacity 10^5
- All models were trained on 100 episodes
- Metrics: Total number of trips, capacity utilization reward (U), Customers served per trip, Negative trip reward (L), Negative distance reward (D), Sum of rewards.

Limitations and Future Directions

- We could not use standard libraries like gym (delayed rewards, various interactions using agents)
- SAT solver does not consider time windows.
- Paper does not mention several important hyper-parameters like vehicle speed.
- Ambiguities in the algorithms: how heuristics are used in the graph generation, and clustering algorithm results in 1 cluster.
- Since we ran it for lesser episodes, the hyper-parameters mentioned did not result in the optimal possibilities.
- Future directions
  - Idling time can easily be incorporated, where vehicles can wait in between a route. This can increase efficiency since vehicles will not keep returning to warehouse.
  - It is also easy to incorporate variable vehicle capacities.

Results H+H

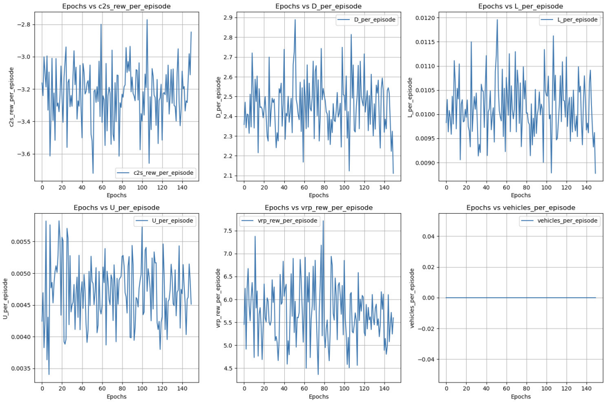


Figure: H+H

Results H+L

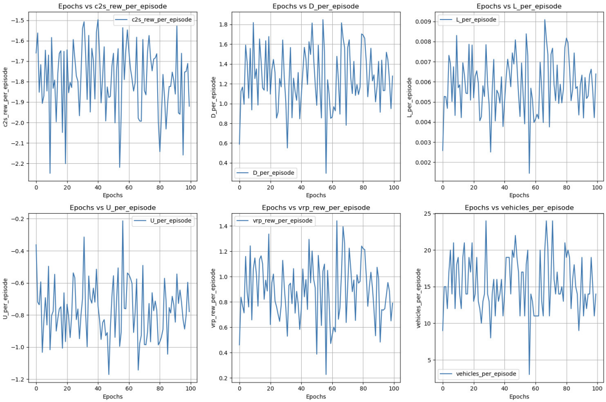


Figure: H+L

Results- L+H

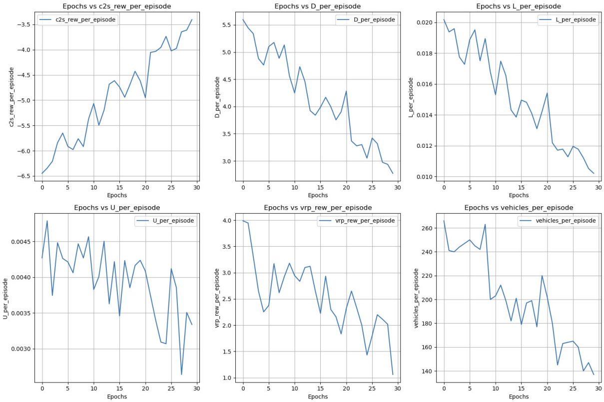


Figure: L+H

Results- L+L

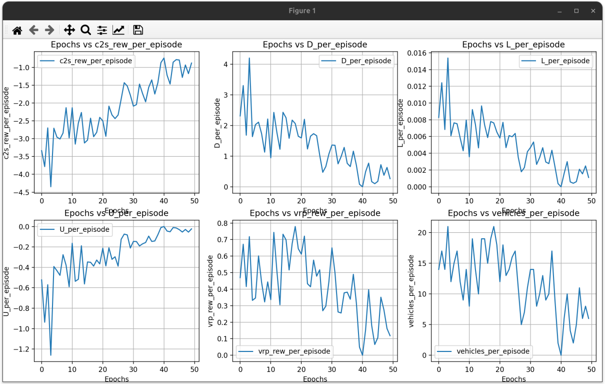


Figure: L+L

Results - all

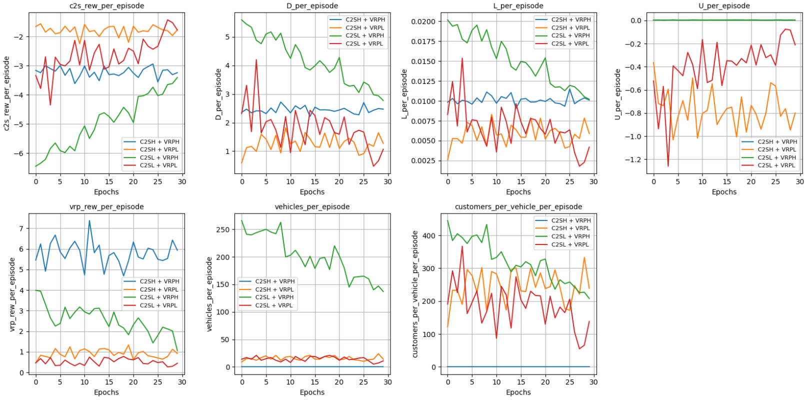


Figure: ALL