CS 285 Project Environment

Fall 2020 Project Report

Ilgin Dogan & Yiduo Huang

1 MARL Environment

Consider a city with many restaurants and drivers. For each driver i, at time t, let U_t^i be the set of orders on board which has already been picked. Let $order_{i,pend}^t$ be the pending order (order to be picked) for driver i.

Algorithm 1: Driver simulation at time t

```
for driver i=1,2,...,N do
    Check if the driver i can pickup pending order or drop some orders;
    if Can drop orders then
        drop orders, update U_t^i;
    end
    if Can pick orders then
        pick orders, update U_t^i;
    end
    Assign an dispatch action a_t^i to driver i;
    if a_t^i corresponds to an new order or dispatching to new areas then
        Calculate the reward for accepting the new order (and give up current pending order order_{i,pend}^t);
        Update pending order order_{i,pend}^t;
    else
        Calculate reward;
    end
    Move according to the updated trajectory;
end
```

We refer to our problem as Online Order-Dispatching and Multi-Driver Repositioning Problem (Online-ODMDRP). It can be formulated as a Partially-Observable Markov Decision Process (POMDP) characterized by the tuple $(\mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{T}, \mathcal{R}, N)$ whose components are described below in detail.

- N: Number of active heterogeneous drivers/agents in the current system
- S: Environment state space that is not fully observable to the agents. We denote the system state at time t as $s_t \in \mathcal{S}$ and it consists of $s_t = (d_t^1, \ldots, d_t^N, C_t, s_t^{\text{city}})$ where d_t^i is the state (i.e. feature) vector for agent i at time t, C_t is the current buffer of unassigned customer orders at time t, and s_t^{city} is the current state of the city (i.e. traffic density) at time t which is assumed to evolve independent from the current states of the agents.
  - Each agent i is characterized by the tuple $d_t^i = (x_t^i, v^i, a_t^i, h^i)$ where
    - x_t^i is the current location of agent i at time t
    - v^i is the vehicle info of agent i, $v^i = [speed_i, available\ seats_i]$
    - a_t^i indicates if the driver has unpicked orders and the fee of this unpicked order.

* κ_t^i is the optimal path to fulfill current orders on board. Let U_t^i be the set of order on vehicle i at time t, $\kappa_t^i := [x(u_{tj}^i), y(u_{tj}^i), t(u_{tj}^i), j = 1, 2, ..., |U_t^i|]$ where $x(u_{tj}^i), y(u_{tj}^i)$ is the drop location of the order u_{tj}^i and $t(u_{tj}^i)$ is the drop time. To make sure that the length of input is fixed, we pad this vector with the last drop location and time on the current trajectory so that $dim(\kappa_t^i)$ = capacity of the vehicle i.

- Each customer order k in C_t is defined as $c^k = (x_{pickup}^k, x_{destination}^k, \rho^k, pt^k)$ where x_{origin}^k is the location of the pick-up point (i.e. restaurant) of order k, the location of the destination point (i.e. customer's location) of order k, ρ^k is the fee that a driver obtains from order k, and pt^k is the latest pick-up time for order k. We assume that if the pick-up time of an order is later than pt^k, then this order is lost and removed from the system.
- s_t^{city} is the feature of the city. For now it is the historical average demand distribution over the 2d city.

- $\mathcal{O} = O^1 \times O^2 \times \ldots \times O^N$: Joint observation space where O^i is the private observation available to agent i. We assume that an agent is not able to observe the state of the other agents. Thus, we define the observation of agent i at time t as $o_t^i = (d_t^i, s_t^{\text{city}}) \in O^i$.
- $\mathcal{A} = A^1 \times A^2 \times \ldots \times A^N$: Joint action space for N agents. The action $a_t^i = \{\zeta_t^i, x_{o,t}^i, x_{d,t}^i, fee_t^i\} \in A^i$ has four components:
  - $\zeta_t^i = 1$ if this action corresponds to taking new orders or dispatching to new areas. $\zeta_t^i = 0$ if this action corresponds to keeping current trajectory.
  - $x_{o,t}^i$ the location that we dispatched this driver to. We dispatched a driver to location $x_{o,t}^i$ if (1) there is a new order at this location. (2) there is no order but the driver could benefit from going to this area before delivering other orders. If $\zeta_t^i = 0$, this should be the first point on the current trajectory.
  - $x_{d,t}^i$ the dropping location of the new order. If this dispatch has no specific order, or $\zeta_t^i = 0$, then $x_{d,t}^i = x_{o,t}^i$
  - fee_t^i is the fee that we earned from the new order. This will be zero if this dispatch has no specific order, or $\zeta_t^i = 0$.
- $\mathcal{R} = R^1 \times R^2 \times \ldots \times R^N$: Joint reward space for N agents. Our reward is inspired from equation (7) in Alabbasi et al. (2019). We compute the reward as follows:

$$r_t^i(a_t^i, d_t^i) = \beta_1 b_{t,n} - \beta_2 c_{t,n} - \beta_3 \sum_{u \in U_t^i} \delta_{t,n,u} + \beta_4 fee_t^i - \beta_5 fee_{t,unpicked}^i \tag{1}$$

where

- $b_{t,n}$ is the number of orders on board.
- $c_{t,n}$ is time taken to go to the dispatched area.
- $\delta_{t,n,u}$ is the additional time vehicle takes before drop order u. If we are going to take a detour, the tsp solution will be changed and it takes more time to drop the order on board. $\delta_{t,n,u} = 0$ if we stick to the old trajectory.
- fee_t^i is the fee we can earned from accepting new orders. $fee_t^i = 0$ if we don't know the new order.
- $fee_{t,unpicked}^i$ is the fee of lost order. If at time t, we decide to go to pick another order while the current pending order $order_{i,pend}^t$ exists. Then we have to give up this order $order_{i,pend}^t$.

- $\mathcal{T}$: State transition dynamics. At each time point, after assigning the orders to agents and updating the paths of drivers, we update the system state components for agent i according to algorithm 1. The city features are updated according to an independent random process.

2 Neural Network Input Architecture

The input of the network $\pi(a_t^i|d_t^i, s_t^{city})$, where d_t^i has the following structure (see comments of function *Driver* :: *agent_observe*)

```
[current_x, current_y,                                  #x_i^t
 speed, available seats,                                  #v^i
 has_unpicked_order, order_to_pick_fee,                   #a_t^i
 trajectory_x0, trajectory_y0, trajectory_t0 - current_time,
 trajectory_x1, trajectory_y1, trajectory_t1 - current_time,
 ...
 trajectory_xm, trajectory_ym, trajectory_tm - current_time  # kappa_t^i
 ]
 If the vehicle has a max capacity of C, then m = C
```

and s_t^{city} is a map of current demand defined as

```
City::request_map
```

The action a_t^i has the following structure (see comments of *Driver* :: *take_action*)

```
[zeta, x0, y0, x1, y1, fee]
zeta = 1 if we will pick (potential) new passengers, =0 otherwise
x0, y0 = location of dispatching
x1, y1 = if we will pick an order in (x0,y0), (x1,y1) will be its destination
if we are not going to pick new users,
x0 y0 and x1 y1 will be the next order drop location
```

3 Note and suggestions

Note: discrete choice on continuous space. The setting is not entirely the same as 'shared experience actor-critic'. Our action space is continuous. But we can only select from a finite set of actions (correspond to new orders and dispatch location). We need to take a softmax selector just like Li et al. (2019). (Our π here is equivalent to μ in equation (6) of paper Li et al. (2019)). I think we can plug in equation (8) in Li et al. (2019) where μ in (8) is our π instead.

Thought 1. go back to continuous space. Another approach is that we simplify the structure of action space. $a_t^i = \{\zeta_t^i, x_{o,t}^i, fee_t^i\}$ and we don't consider 'pending order'. (Only dispatch, no new order). When the driver got to the dispatch point, it automatically picked the order with best reward. In this case, the action space is continuous so we can apply the algorithm directly.

However, using this approach, the drivers may not know the destinations of new orders when dispatched to that area. One solution is to add an 'order drop heatmap' for each restaurant area to s_t^{city}.

Thought 2. Continuous or discrete?. If the grid size is small, all continuous variables can be approximated by integer. Maybe we can

Thought 3. Some possible network structures. *zeta* is a Boolean variable. We might need to think about special network structure (for example, two independent NN, one for zeta=0 another for zeta=1).

$$\pi(a_t^i|d_t^i, s_t^{city}) = \zeta_t^i \pi_1(a_t^i|d_t^i, s_t^{city}) + (1 - \zeta_t^i)\pi_0(d_t^i, s_t^{city})$$

In addition, if $zeta = 0$, we actually don't need to consider $x0, y0, x1, y1, fee$, so this network is $\pi_0(d_t^i, s_t^{city})$ (no input about action).

For the input s_t^{city} which is a 2d matrix representing demand distribution. We may need a GNN (If we are going to use sparse graph instead of full city, where each node representing an restaurant. This is different from carpool setting since location of restaurants are fixed) or CNN structure to deal with the input.

References

Abubakr Alabbasi, Arnob Ghosh, and Vaneet Aggarwa. Deeppool: Distributed model-free algorithm for ride-sharing using deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 20(12):4714–4727, 2019.

Minne Li, Zhiwei (Tony) Qin, Yan Jiao, Yaodong Yang, Zhichen Gong, Jun Wang, Chenxi Wang, Guobin Wu, and Jieping Ye. Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning. In *Proceedings of the 2019 World Wide Web Conference (WWW'19),*. ACM, New York, NY, USA, 2019. doi: https://doi.org/10.1145/3308558.3313433.