Code Description of Team Polar Bear

Description of the Dispatching Agent

We use a TD(0) on-policy algorithm to dispatch the orders. The details are as follow.
The values are stored by two hashmaps, namely grid_values and layer_values, which hash grid ids provided by the competition to numerical values. Grid_values record the value of a certain grid while layer_values are values of grids around a certain grid. All the values are initialized by 0.
For each driver-order pair, we calculate the expected update by:

$$U = (1 - p) \cdot (r \ + \ \gamma^t V' - V), \text{ where}$$

p is the estimated cancel probability: $p = 0.01 \cdot \exp\left(\frac{\log(20)}{2000} \cdot d\right)$, d is the order_driver_distance;

r is the reward of an order;
γ^t is the discount factor, $t = (order\_finish\_time - order\_start\_time)/600$;
V' is the average of grid_value and layer_values corresponding to the grid where the destination of the order is located;
V is the average of grid_value and layer_values corresponding to the grid where the driver is located.
Then we use KM algorithm to make assignments between drivers and orders where the weight between a driver and an order is the expected update U above. Before applying the KM algorithm, we prune the drivers too far away from an order and only leave top-k nearest drivers of an order where k is set to 11.
According to the assignment results, we update grid_values and layer_values respectively. If a driver is successfully assigned to an order, the values will be updated by adding $\alpha \cdot U$ where α is the learning rate. Otherwise, i.e., the driver remains idle, the values will be updated by adding $\alpha \cdot (\gamma - 1) \cdot V$.
We do not use any pre-trained models. The KM algorithm is implemented by C++ and we call the function with a dynamic link library “hungnp.so”.

Description of the Repositioning Agent

It is the default repositioning agent that leaves the drivers staying where they are. No RL techniques have been applied.