Abstract

Last-mile delivery is a rapidly evolving field. Every company specializing in this area faces the challenge of assigning couriers to customer orders. The main question is how to make these assignments most efficient.

In this thesis, I formally posed this problem and, using real-world simulations, explored the possibilities of solving it with neural networks via reinforcement learning. The resulting algorithm is compared with several heuristic algorithms to evaluate its performance relative to a baseline.

This work has several outcomes. Firstly, I implemented an infrastructure that allows for simulating couriers and orders; I developed the neural network architecture; and I implemented an algorithm that trains models using reinforcement learning. Secondly, I analyzed the model's performance based on various parameters. I tested different reward functions, approaches to working with geo coordinates, and model sizes. Thirdly, I proved several theoretical statements that apply to various similar problems. Specifically, I proved the optimality of one algorithm under certain conditions and proposed a method for dealing with certain types of sparse rewards.

The current conclusion of this work is that for the basic problem of assignment, heuristic algorithms that rely on the distances between couriers and orders are quite effective. Neural networks can learn to solve these problems at a similar level but have not yet succeeded in improving the results.

Thus, this work provides only an initial understanding of the problem and its possible solutions, leaving several questions for future research. For example, the presented model did not cover the possibility of adding a new order to an existing one, although the necessary functionality is implemented in the code. This could be one of the areas where reinforcement learning might demonstrate higher performance compared to heuristics.

Content

1 Introduction 3

1.1 Last-mile delivery problem 3

1.2 Related works 3

2 The dispatch problem 4

2.1 General setup 4

2.2 The model of environment 4

2.3 Metrics 6

2.3.1 Complete Rate (CR) 6

2.3.2 Click to Delivery (CTD) 6

2.3.3 Arrival distance 6

2.4 The discrete-time dispatch problem 7

2.5 Application in the real world 7

3 Dispatches 8

3.1 Heuristic dispatches 8

3.1.1 Random dispatch 8

3.1.2 Greedy dispatch 8

3.1.3 Hungarian dispatch 9

3.2 Neural network dispatches 9

4 Reinforcement learning 10

4.1 Reasoning 10

4.2 Reinforcement learning setup 10

4.2.1 State 11

4.2.2 Action 11

4.2.3 Reward 11

4.3 Algorithm 13

4.4 Neural network . . . 13
4.4.1 State encoder . . . 14
4.4.2 Attention . . . 14
4.4.3 Actor-critic heads . . . 14

5 Experiments **15**
5.1 Data . . . 15
5.2 Behaviour cloning . . . 16
5.2.1 Easy data . . . 16
5.2.2 Hard data . . . 17
5.2.3 Results . . . 18
5.3 Stability problems . . . 18
5.3.1 Examples of instability . . . 18
5.3.2 The form of loss . . . 19
5.3.3 Instability hypotheses . . . 20
5.4 Reward function . . . 20
5.5 Importance of distances . . . 21
5.6 Length of training . . . 22
5.6.1 Short training . . . 22
5.6.2 Long training . . . 23
5.7 Results . . . 24

6 Conclusion **25**

A Hungarian dispatch optimum **26**

B Encoder architecture **27**

C Densified reward **28**

D Working with coordinates **29**

1 Introduction

1.1 Last-mile delivery problem

Every last-mile delivery company solves the problem of courier scheduling. The concept of last-mile delivery is not very complicated. The goal is to maximize the number of delivered orders in the fastest way. The problem is that we want to minimize the number of courier hours because their costs reduce our profits. That's why the question is how to find the most efficient way in which one can schedule couriers (supply) on claims (demand) in a dynamic system.

In this work, I focus on one of the possible formulations of the last-mile delivery problem. I will formally describe the problem, introduce some heuristic methods, and then apply reinforcement learning to solve it with neural networks.

1.2 Related works

Even though many logistics companies solve the same problem and maximize similar metrics, there are a lot of different formulations of last-mile assignment problems that focus on different details. That's why most parts of the existing research are weakly connected with this work.

Thus, some papers [2] [10] are focused on route optimizations, some [3] investigate theoretically how to balance supply and demand amounts, and some of them [4] research heuristics of effective assignments according to geography.

Reinforcement learning (RL) is not a new approach for such tasks. For example, one research [8] investigates order insertions. Authors allow couriers to have more than one package. They train RL agent who inserts new orders into others effectively. Other researchers focus on route optimization via RL [6] [7].

The most relevant [5] research to my work studies the same question as me. The authors have almost the same streaming setup and optimize similar metrics. The main difference is that they do not compare their model with any baselines.

They only show that the problem could be solved with RL.

Another relevant research [9] was made using real data. The authors train an RL agent to predict where to carry the package. Moreover, they compare the results with some baseline algorithms and compute similar metrics. The main difference from my work is that the authors focus on the multihop trajectory of the package in the city rather than assignments on couriers.

2 The dispatch problem

2.1 General setup

Consider a stream of claims - requests from clients to deliver their packages. A claim could be canceled if it is not assigned for a while. Consider a second stream of couriers. Every courier starts work at some time and he is available until some end time.

Dispatch D is a function that takes these 2 streams and produces an output stream of assignments online. The assigned courier delivers a given claim, and after that, the claim is considered completed.

We aim to find dispatch that maximizes given metric Q on a stream. More formally we should maximize $Q_* = \lim_{time \to \infty} Q(t)$ over D, where $Q(t)$ is a metric Q computed on a stream of length t. Usually, it is enough to maximize $Q(t)$ for some big t. We discuss some examples of Q later.

2.2 The model of environment

In a general scenario, couriers and claims are very general objects. Consider a specific model with which we would work. Let's describe this model.

Consider a **system** S, which contains the following objects.

Courier is a tuple of dynamic geo-point (position), start time and end time, and speed.

Claim is a tuple of 2 geo-points (source and destination), created time, and cancel time (if not assigned).

Route is a dynamic curve on a plane. We will consider routes defined by a list of points that should be visited in a specific order.

Order is a tuple of a courier, a route, and a claim (generally, a set of claims if we allow one courier to carry several claims). The order is treated as completed when all the claims within are completed.

Assignment at time t is a set of pairs (courier, claim). Pair (A, B) means that courier A should deliver claim B.

Our dynamic system $S(D, P)$ is determined by dispatch D and distribution P. Couriers and claims with all their attributes are sampled from P.

The dynamic of the system S is defined in the following way. For every time moment t:

1 Couriers with start time t are initialized in S

2 Claims with created time t are initialized in S

3 Couriers that are not in orders and that have an end time greater than t are removed from S

4 Claims that are not in orders and that have a canceled time more than t are removed from S

5 Orders are formed according to assignments from D and are initialized in S

6 In every order, couriers are moved through their route with respect to their speed

7 If the destination point of a claim in order O is visited by a courier in order O, then the claim is treated as completed and removed from S. If the route of order O if empty, O is removed from S, however, the courier from O still stay in S

2.3 Metrics

The most obvious metric that could be optimized is profit, which is considered an objective in many related works. However, profit depends on prices, which are business-specific parameters. That's why, in this work, I consider more general metrics.

2.3.1 Complete Rate (CR)

CR is the main metric that shows the proportion of completed claims among all created ones. It is the second important metric after profit. The problem with this metric is that it is volatile, and we will see this in experiments.

2.3.2 Click to Delivery (CTD)

CTD is computed among completed claims and represents the average time of claim delivery. Time of claim delivery is a difference between two moments: when the claim is completed and when the claim is created. As we saw in the experiments, in this model, CTD correlates very much with CR; however, CTD is less volatile.

2.3.3 Arrival distance

Arrival distance is the average distance between courier A and the source point of claim B computed among all assignments (A, B). It is a technical metric that approximates the effectiveness of dispatch and could be optimized directly.

Usually, arrival distance correlates with another important metric: estimated time of arrival (ETA). ETA is also a very important business metric.

2.4 The discrete-time dispatch problem

Here, I want to formulate the exact problem I want to focus on. Previously formulated continuous-time problem is not applied in practice. Usually, one wants to work with discrete-time problems. There are several reasons for that.

Firstly, many side microservices (for instance, the one that returns actual courier coordinates) are much more efficient in batch mode.

Secondly, any dispatch executes for some significant time, no matter how effective.

Thirdly, errors in computing the attributes of couriers, as well as network delays, make the stream manner of the problem meaningless.

All these reasons encourage us to consider a sequence of sets rather than a stream of items. That's why we consider the problem with discrete time.

Let δt be a sampling rate. The definition of system dynamic remains almost the same. The only difference we need to make is to replace time moment t with time interval $[t; t + \delta t)$.

Gamble at time t in a discrete-time system S is a tuple of couriers, claims and orders in S at time t.

Dispatch in a discrete-time system S is a function which takes gamble and returns assignments for time interval $[t; t + \delta t)$.

I will consider this formal problem for the rest of this work.

2.5 Application in the real world

It is worth noting that in real life, the complexity of the task is not so much in finding an algorithm to solve the problem and not in the performance or accuracy of this algorithm but in additional local constraints of the problem and edge cases arising from the business requirements. Such restrictions may be prioritization of clients (whose applications need to be delivered faster) or incompatibilities between some attributes of couriers and claims (for example, foot couriers should not carry heavy claims).

Thus, well-functioning heuristics become overgrown with many local constraints, which become difficult to manage. In addition, all theoretical guarantees of algorithms are lost, making it completely unclear how far from the optimum the algorithm works.

3 Dispatches

3.1 Heuristic dispatches

Here are some examples of heuristic dispatches, which would be our baselines in this work.

3.1.1 Random dispatch

Consider a random dispatch as a zero-point baseline. For every claim we sequentially choose a random available courier.

3.1.2 Greedy dispatch

For every claim we sequentially choose the best available courier. Here, we need a quality metric $Q(A, B)$, which shows how courier A fits claim B. I consider $Q(A, B) = e^{-d(A,B)}$, where $d(A, B)$ is the distance between courier A and the source point of claim B.

.

3.1.3 Hungarian dispatch

This dispatch is not sequential and uses all the information about claims and couriers.

Consider a bipartite graph where couriers are nodes in the first component, and claims are nodes in the second component. The weight of an edge between courier A and claim B is $Q(A, B)$. One can use the Hungarian algorithm [13] to find a maximum matching. Edges in this matching are the assignments of the dispatch.

This dispatch locally minimizes the distances between couriers and claims. Under some conditions, it is the optimal dispatch (see Appendix).

It's worth mentioning that the Hungarian algorithm has $O(n^3)$ complexity, which is fine in most cases.

3.2 Neural network dispatches

The core of this dispatch is a neural network (NN). There are several ways to implement an NN-based dispatch.

I consider general architecture. Given a claim, N couriers, and any additional information, the NN should return a vector of logits of length $N + 1$. The logits should be non-zero and sum up to 1 (probability distribution). The last logit corresponds to a fake courier. The claim which is assigned to a fake courier is not assigned at all.

Given this NN, one can sequentially iterate through all the claims and choose the most probable courier according to output logits.

The description of the NN architecture I used is described in 4.4.

4 Reinforcement learning

4.1 Reasoning

The use of the RL approach is due to the inability to determine the target in the case of supervised model training mode. It turns out that any attempts to somehow determine the “correct” assignments for this fail on the definition of a metric $Q(A, B)$ that shows how claim A fits courier B. With such a metric, we can easily run Hungarian dispatch, optimizing $Q(A, B)$ directly.

I suggest we accept the fact that we do not know what the “correct” assignments are and delegate the understanding of this phenomenon to the RL agent. We additionally get a few more bonuses from this approach.

Firstly, the most important metrics for a business are global for the entire system S (for example, CR or CTD). These metrics cannot be optimized locally in gamble (unlike arrival distance). In the RL approach, these metrics can be declared a reward and maximized directly.

Secondly, any additional local constraint is a great difficulty for a heuristic algorithm, often accompanied by incomprehensible effects in algorithm performance. For example, it is unclear which parameters need to be changed to reduce the CTD for claims of a specific client. In RL, all these difficulties are the agent’s concern. Restrictions need to be implemented, and penalties added to the reward.

That’s why the core idea of this work is to train an NN-based dispatch as an RL agent.

4.2 Reinforcement learning setup

Let’s define some RL objects here.

4.2.1 State

The straightforward idea is to trait gamble as a state. However, in this approach, states and actions are sequences. Despite it being possible to work with sequences, I decided to use a simpler way. Let's consider every claim in the gamble as a state. Let me formalize this idea.

Consider **state** as a triple (G_t, i, A), where G_t - is a gamble at time step t, i is an index of a current claim in G_t, A is a current set of assignments for G_t which may be updated. Let's say that the state is **transitional** if i is the index of the last claim in G_t.

Intuitively, we want to return assignments A as a dispatch output only in transitional states.

4.2.2 Action

Action is either an index of courier in G_t, which we want to assign to the given claim, or -1 if we don't want to assign this claim to any courier. Action is transitional if and only if the state in which the action is made is transitional.

In the case of non-transitional action a_i (associated with a claim of index i), the state is changed in a deterministic way:

$(G_t, i, A) \xrightarrow{a_i} (G_t, i + 1, A \cup \{a_i\})$

In the case of transitional action, we return assignments A and receive a new gamble:

$(G_t, i, A) \xrightarrow{a_i} (G_{t+1}, 0, \emptyset)$

4.2.3 Reward

The reward may be non-zero only on transitional actions because only in these cases does the interaction with the environment occur.

Reward function $R(G_t, A_t, G_{t+1})$ should be searched experimentally. There are a lot of ways how to create such a function [15]. In my work, I consider 3 types of reward functions that would be compared in experiments.

1 **Sparse reward** which is defined as CR for the last f steps if $f = tk$ for some k. For other t, the reward is zero.

   It is the most intuitive reward - direct maximization of target metric on some time period. The problem with this reward is its sparsity. However, sometimes [16] sparse reward may work better than a dense one.

2 **Additive reward**. Let $R(G_t, A_t, G_{t+1}) = \sum_j \beta_j n_j$, where n_j are some statistics calculated on A_t, G_t or G_{t+1}. Betas are some coefficients.

   For example, n_1 could be the amount of non-fake and valid assignments in A_t, and n_2 could be the amount of new claims created in time interval $[t, t+1)$.

   Although this reward is very flexible, finding the appropriate coefficients could be challenging. Moreover, there is no guarantee that the reward will maximize target metrics.

3 **Densified reward**. A lot of business metrics have a form $M = \frac{\sum_{\tau=0}^{t} m_\tau}{\sum_{\tau=0}^{t} n_\tau}$, where n_t is the amount of created claims at time step t and m_t is any statistic computed at time step t. CR is a good example.

   The problem with such metrics is that they could be computed just at the end of the period, i.e., they are sparse. Luckily, it is possible to make M dense (densify). The formula is derived in Appendix.
   Here, I provide the final formula of densified CR, where m_t is the number of completed claims at the time interval $[t; t+1)$.

   Let $M_t = \sum_{\tau=0}^{t} m_\tau$ and $N_t = \sum_{\tau=0}^{t} n_\tau$ - cumulative amounts.

Then:

$$R(t) = \begin{cases} \frac{m_t N_{t-1} - n_t M_{t-1}}{(N_{t-1}+n_t)N_{t-1}}, & \text{if } N_{t-1} \neq 0 \\ \frac{m_t}{n_t}, & \text{if } N_{t-1} = 0 \text{ and } n_t \neq 0 \\ 0, & \text{else} \end{cases}$$

It can be proven that $CR(T) = \sum_{t=0}^{T} R(t)$

That means that maximization of such reward is equivalent to CR maximization.

4.3 Algorithm

Proximal Policy Optimization (PPO) [12] is used as the default RL algorithm. I want to note that it might be a good idea to use off-policy algorithms on baseline trajectories. However, to overtake the baselines, we would still have to implement an on-policy algorithm.

4.4 Neural network

The whole NN architecture consists of 3 parts: state encoder, attention, and actor-critic heads.

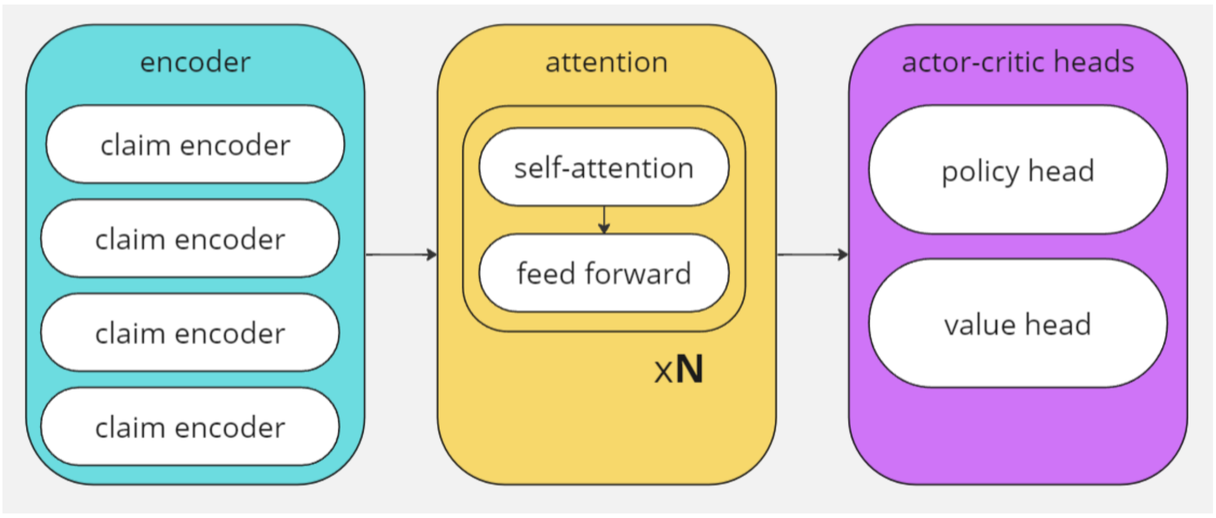


Fig. 4.1: Model architecture

4.4.1 State encoder

The encoder takes the state G_t, i, A as input. The state contains features of couriers, claims, and orders, as well as global features of the gamble G_t (for example, time of day or number of claims), information about current assignments of the model A, and an index of the considered claim i. Each claim, courier, and order with additional information are encoded into vectors (tokens) using appropriate encoders (see Appendix).

A fake courier token and a state-value token are added to this sequence of tokens. A fake courier token is responsible for non-assignment for a given claim. A state-value token will be used in the future to get a value of the state by the critic in the actor-critic model. This sequence of tokens, having the length $n_{crr} + n_{ord} + n_{clm} + 2$ (where n_{crr}, n_{clm}, n_{ord} are the number of couriers, claims, orders in the gamble, respectively) is the output of this part.

4.4.2 Attention

The attention part takes a sequence of tokens as input and applies several layers of self-attention [17] and feed-forward operations (essentially a transformer's [11] encoder). A sequence of tokens of the same length is an output. After this phase, tokens have information about each other.

4.4.3 Actor-critic heads

The actor-critic heads part is just a pair of few-layer MLPs corresponding to the policy head and the value head of an actor-critic model (see PPO).

All the tokens corresponding to couriers are passed through the policy head and softmax. The output is policy logits. The state-value token is passed through the value head. The output is a state value.

5 Experiments

5.1 Data

For training control I use 2 types of data: **easy** and **hard**

Easy data is generated from simple distributions. Thus, all coordinates are sampled from uniform distributions. Couriers and claims have constant lifetimes, every gamble constant amounts of new couriers and claims are sampled.

Due to its simplicity, such synthetic data makes it possible to find problems in algorithms, as well as get a significant difference in quality for similar baselines.

Hard data is generated from distributions that approximate real data distribution in some sense. Claims and couriers are generated with dynamic intensity and different densities (regarding coordinates). Lifetime is sampled from an exponential distribution. Every gamble is sampled random amounts of claims and couriers. The parameters of the distributions were selected by eye based on the author's experience and common sense.

Due to their randomness, such data have many heterogeneities (for example, there are many claims during rush hour and a lot of free couriers during quiet times). All these problems are inherent to real data. It is more difficult for models to train on this data, and the results of some baselines do not show significant differences.

Estimating model performance by computing CR for the whole day is too long. I use sampling rate $\delta t = 30$ seconds, so 2880 simulator steps should be made to compute daily metrics.

That's why I compute all metrics for 500 steps (about 4 hours). Experiments showed that it is enough in terms of significance.

5.2 Behaviour cloning

RL training often is a complicated process that requires searching all the PPO parameters (for example, coefficients in GAE [18]) and model hyperparameters accurately. That's why it would be a good idea to measure the model's performance in supervised mode. Moreover, in this mode, we will get a lower bound of NN-based dispatch quality.

Let's take any of our baselines and encourage our model to make the same assignments. The baseline is used for trajectory sampling as well. I use cross-entropy loss, where the target is an index of the courier selected by baseline for a given claim.

5.2.1 Easy data

Table. 5.1: Easy data. Behavior cloning results

Model	CR	CTD	Arrival-distance
medium-Hungarian	0.845	2465*	0.031***
medium-greedy	0.846	2467***	0.031***
big-Hungarian	0.847	2471	0.031**
tiny-Hungarian	0.846	2531***	0.036***
hungarian	0.846	2447	0.028
greedy	0.802	2879	0.089
random	0.736	3845	0.220

The model with the name "a-b"means that the model has size "a"and was trained to clone baseline "b". "***"means a significant difference between the model results and the Hungarian baseline. The baselines are marked in bold.

All the baselines significantly differ from each other (in all 3 metrics). In these experiments, all the considered models are significantly better than greedy and random baselines.

There are some conclusions from these experiments:

1 All the models (even tiny) have enough capacity to perform as Hungarian baseline in CR.

2 Models fail to minimize distance as well as the Hungarian baseline does.

3 The size matters. Big and medium models achieve the same CTD as the baseline, while the tiny model does not.

4 The model trained to clone the greedy baseline performs better than this baseline.

5.2.2 Hard data

On the hard data, greedy and Hungarian dispatches give insignificant results for all 3 metrics in a period of 500 steps. In the horizon of the whole day (2880 steps) some metrics become significant, however, it is too long to compute them. That's why I will train the model to clone only the Hungarian baseline.

Table. 5.2: Hard data. Behavior cloning results

Model	CR	CTD	Arrival-distance
medium-Hungarian	0.632	2546	0.043
big-Hungarian	0.596	2298***	0.018***
hungarian	0.688	2639	0.038
greedy	0.662	2674	0.041
random	0.498	3956	0.183

We can see that the big model could even outperform the baseline it was trained to clone (in terms of CTD and arrival distance).

5.2.3 Results

Now we know that models have enough capacity to perform at least the same as the best of our baselines (and sometimes even better). Unlike baselines, our models may not assign a claim even if there are available couriers. Moreover, our models can use additional information about claims and couriers. The absence of these restrictions may allow our models to beat our baselines significantly.

5.3 Stability problems

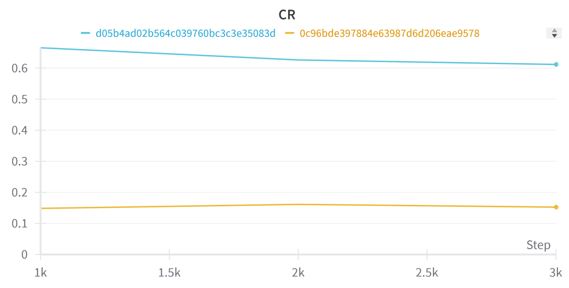
The main obstacle in my work was the instability of RL training.
Even after the best PPO parameters and training hyperparameters were chosen, training on both easy and hard data often failed. Even though the learning rate was really low (about 10^{-5}) and a one-cycle scheduler was used, the problems remained.

5.3.1 Examples of instability

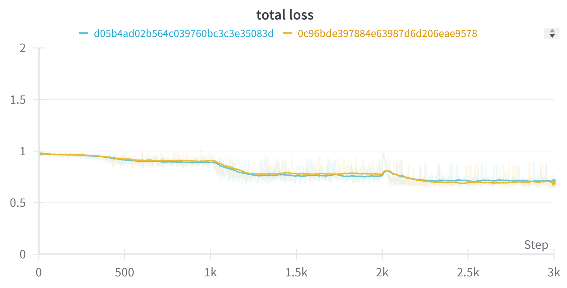
The instability manifested itself in different ways.

Firstly, equivalent runs with the same parameters may either give completely different results (better than the best baseline and worse than the random baseline) or diverge at all. The example of this phenomenon even with the similar loss in demonstrated at 5.1.

Secondly, there was an often situation when quality metrics dropped dramatically when training even on very small learning rates. In figure 5.2 CR drops from 0.84 (which is about the highest result) to 0, while the loss looks fine. In figure 5.3 CR suddenly drops from 0.79 (which is a greedy dispatch quality) to 0.65 (which is worse than random dispatch quality).

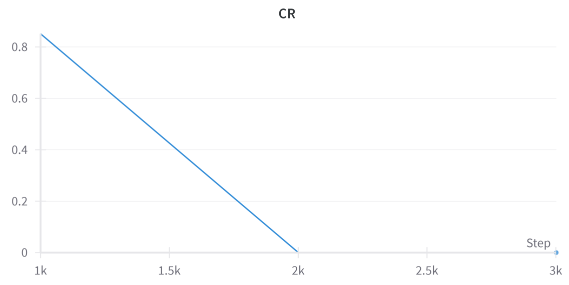


(a) CR of 2 runs

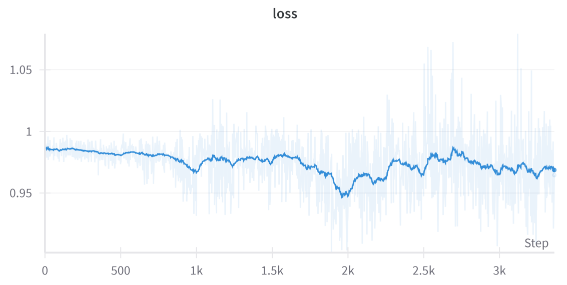


(b) losses of 2 runs

Fig. 5.1: Successful and unsuccessful runs with the same training parameters

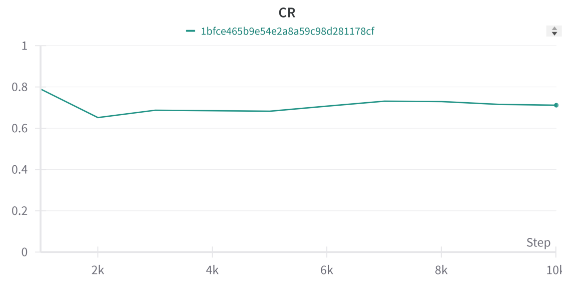


(a) CR

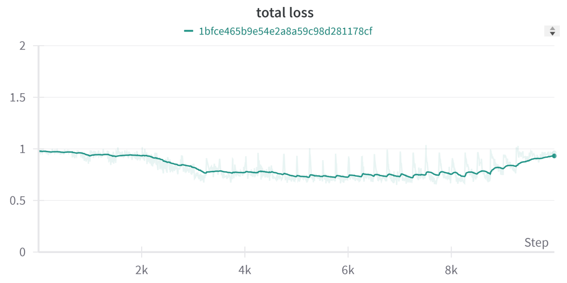


(b) loss

Fig. 5.2: A drop of CR while the loss looks fine



(a) CR

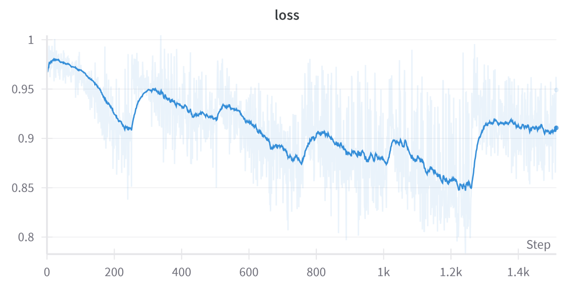


(b) loss

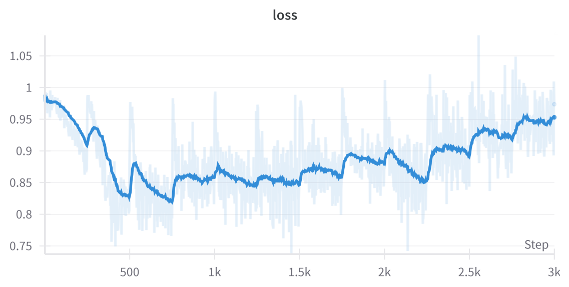
Fig. 5.3: Sudden drop of CR at the beginning

5.3.2 The form of loss

The reason for these instabilities may be covered in the loss behavior. There are two interesting characteristics of the loss on this task no matter whether the run was successful or not. The examples are demonstrated at 5.4. Firstly, the losses are periodic - this may be explained by trajectory resampling. However, it is unclear, why in these losses the periods are so pronounced. Secondly, one may see the losses have an upped trend. Despite the fact that at the end of training the learning rates are too low (about 10^{-7}).



(a) loss (example 1)



(b) loss (example 2)

Fig. 5.4: Periodic growing up loss

5.3.3 Instability hypotheses

I found several hypotheses that can explain this phenomenon.

1 This may look like an overfit on sampled trajectories.

2 Stomping in the local optima with low learning rate.

To check some of these hypotheses, I ran many experiments with different learning rates (in the range $[10^{-7}; 10^{-2}]$) with and without schedulers and other learning control parameters. However, it looks like the problem remains no matter what the learning rate is and other parameters are. I've tried different regularizations like dropout or weight decay. However, this did not help.

The only thing I came up with to deal with this problem is running several runs with the same parameters. That's why, in further experiments, I will report only the best run of 2-3 runs.

5.4 Reward function

I tested three types of reward functions: "sparse reward,additive reward,"and "densified reward."I ran models for 5k and 10k training iterations.

Table. 5.3: Easy data. Reward functions experiments

Reward function	CR	CTD	Arrival-distance
dense-AR	0.849	2688	0.053
dense-AR (long)	0.816	2467	0.032
dense-CR (long)	0.835	2675	0.044
sparse-AR (10)	0.838	2354	0.019
sparse-AR (30)	0.618	4784	0.343
sparse-AR (90)	0.822	2954	0.078
Additive	0.704	4013	0.246
Additive (new claims)	0.818	2980	0.097
Additive (dist)	0.81	3068	0.109

Here 5.3 reward function "a-b (c)"means that a reward function "a"with metric "b"uses the parameters "c".
It is important to notice, that there is no significant difference between algorithms with result CR in the interval $[0.80; 0.84]$. All these models beat greedy dispatch and lose the Hungarian dispatch.

The most stable reward, which converged almost always, is an additive reward with the formula $R = n_{assigned} - avg(\text{arrival distance})$. However, the target of the model with such a reward function is, in fact, the minimization of arrival distance. Our baselines optimize the same metric. That's why this case is not really interesting - we want to maximize CR directly, not minimize strongly correlated metric.

5.5 Importance of distances

In this section, I investigate what the reason for similar quality of RL approach and baselines.

All our baseline algorithms use information about arrival distances between couriers and claim source points. By default, I also provide this information as interaction features to an NN dispatch. However, it is not obvious if a trained NN dispatch can achieve the same high quality without using this information. Moreover, maybe the NN dispatch doesn't even use any information except the arrival distances. This hypothesis may explain the similarity of qualities.

Table. 5.4: Easy data. Distance feature importance

Model	CR	CTD	Arrival-distance
default	0.849	2688	0.05
without distances	0.736	3817	0.215
only distances	0.845	2899	0.076
hungarian	0.846	2447	0.028

One can see that our models do not use any features but distances on easy data 5.4.

5.6 Length of training

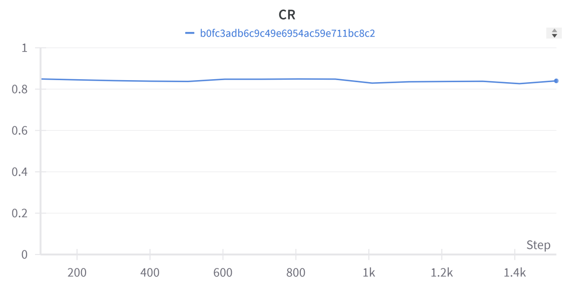
In this chapter, I give some analysis about the length of training. In all these experiments, optimal hyperparameters are used, and "dense reward (AR)"(where "AR"stands for "assignment rate this metric is densified) is used as a reward function.

5.6.1 Short training

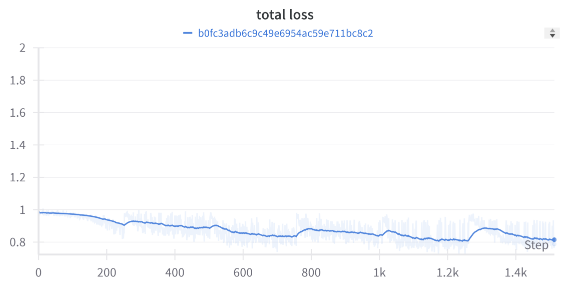
The idea is to see what happens with quality metrics in the first 1500 training iterations. Trajectories update (resample) every 250 iterations.

One of the best runs gives us the result, which differs insignificantly from the Hungarian baseline 5.5.

One can see that CR achieves its maximum (0.849) after the first 100 iterations. That means that the model achieves the best result, in fact, in off-policy mode. This particular case further does not affect the quality of the model at all.



(a) CR



(b) loss

Fig. 5.5: The most success run (with highest CR)

I don't know the reasons for such a phenomenon, but it looks weird that the model converges to the best parameters on a single trajectory in 100 iterations.

5.6.2 Long training

There might be a case that in previous scenarios the model converged to a local optima, and we need to train it for a while to achieve a global optima.

I ran training 30k iterations and compared the results with "short training". It does not seem that models really benefit from "long training"(see 5.5).

Table. 5.5: Length of training

Model	CR	CTD	Arrival-distance
5k (easy)	0.849	2688	0.052
30k (easy) distances	0.37	3506	0.161
hungarian (easy)	0.846	2447	0.028
greedy (easy)	0.802	2879	0.089
random (easy)	0.736	3845	0.220
5k (hard)	0.534	2870	0.10
30k (hard) distances	0.474	3480	0.151
hungarian (hard)	0.624	2516	0.044
greedy (hard)	0.603	2536	0.046
random (hard)	0.413	3660	0.182

5.7 Results

The results are disappointing (see 5.6). It seems that heuristics algorithms are very effective at this task. RL algorithms, in turn, are very unstable on this task.

Table. 5.6: The best trained models

Model	CR	CTD	Arrival-distance
Best (easy)	0.849	2688	0.052
hungarian (easy)	0.846	2447	0.028
greedy (easy)	0.802	2879	0.089
random (easy)	0.736	3845	0.220
Best (hard)	0.534	2870	0.10
hungarian (hard)	0.624	2516	0.044
greedy (hard)	0.603	2536	0.046
random (hard)	0.413	3660	0.182

On the easy data task, the quality of the best baseline was almost achieved (it was achieved in terms of CR and CTD, but not in arrival distance). Moreover, it worked out in the first few hundred iterations, that is, in fact, in off-policy mode. Apparently, the model has learned to rely on the pairwise distances between couriers and claims, as our baselines do.

Unfortunately, on the hard data, it didn't work out to significantly beat even a random baseline, using CR maximization. That is, the model still learned some valid assignments, but due to the large heterogeneities in the data, it could not even learn how to assign applications to the nearest couriers. Although to be fair, I've done quite a few launches on the hard data. However, the random baseline was beaten using a distance minimization reward.

It turns out that in the assignment dispatch problem, heuristic algorithms based on pairwise distances between claims and couriers are quite strong baselines. Some additional information does not help neural networks beat these baselines. Moreover, neural networks in the RL approach learn very poorly and are unstable on this task.

6 Conclusion

In this work, the last-mile dispatch problem was investigated.

I formulated the problem mathematically. Then, I focused on its specific application and implemented it (the simulator) in the code. Then, several simple baselines were described and investigated. After that, the task of RL training using neural networks was formulated. Many experiments were conducted that estimate various reward functions and test different hypotheses.

As a result, I failed to beat heuristic baselines using the RL approach. In some cases, it was possible to achieve the quality of the best of baselines by beating the greedy algorithm. In more realistic data, I failed to beat the algorithms of random assignments. Moreover, RL training on this task turned out to be very unstable.

To the best of my knowledge, my work is one of the first in this field. The dispatch task was comprehensively investigated (including assignment algorithms, simulations, and different types of data), heuristic baselines were proposed, and RL approach was compared with them.

In the future, there is a lot to do to improve this work. Firstly, I had very limited resources, so I could not make enough runs to estimate the real quality of the RL algorithms. Secondly, there were shown some phenomenons in the training process which I can't explain. A low quality of training may be a consequence of these phenomena. Thirdly, the feature of inserting claims to an existing order remained unexplored. Both the architecture of the neural network and the implementation in the code allow the researcher to investigate this feature. I believe it is possible to get better results and beat heuristic algorithms in further work.

The investigation of the dispatch assignment problem is very important for last-mile delivery. I think that his work makes a significant contribution to the research of the field, although the results clearly have the potential for improvement.

A Hungarian dispatch optimum

Consider a family of dispatches such that for any dispatch D in the family, the following conditions hold:

1 D uses only the information about claims' source points and couriers' positions

2 D makes maximum possible assignments in a gamble

Under these restrictions, Hungarian dispatch is the optimal one in this family in terms of CR.

Proof

Consider CR metric on some big time horizon $Time$.
$CR = \frac{\#[\text{completed claims}]}{\#[\text{created claims}]} \xrightarrow{time \to \inf} \frac{\#[\text{assigned claims}]}{\#[\text{created claims}]} = \frac{\#[\text{assigned claims}]}{const}$

So, we need to maximize #[assigned claims] over D, where D is a dispatch.

Let D_* be an optimal dispatch and N_* be a #[assigned claims] of D_* on time a fixed time horizon $Time$. Let $T(D)$ be a time on which dispatch D achieves #assigned claims $= N_*$. $T(D) \geq Time$ and $T(D_*) = Time$
So, our goal is to minimize $\mathbb{E}_P T(D) \overset{D}{\to} min$, where P is a distribution of a system.

Consider any time step t. Because D doesn't see destination points, any assignment is interpreted as vanishing of assigned courier for time $\frac{1}{v}$([arrival distance to clai
[distance of claim]). Then, the courier appears again at any random point. So, the only way D can influence the future is by choosing arrival distance to claim for all claims-couriers in the gamble. D can not decide if it wants to assign a claim or not because it needs to make all $min(\#[claims], \#[couriers])$ assignments. Therefore, any t D solves the problem of minimizing the weight of matching in the bipartite graph of claims and couriers. For this problem, the Hungarian algorithm gives the optimal solution.

B Encoder architecture

The encoder consists of 4 different encoders: claim encoder, courier encoder, order encoder, and global encoder.

Claim encoder

A claim is represented as a vector of coordinates (source and destination points) and a vector of other features (distance, time since created, status, etc.). The vector of coordinates is encoder with a point encoder, and the vector of other features is encoded with a few layers of MLP. Then, two vectors are concatenated and passed through another few layers of MLP.

Courier encoder

Courier encoder works in the same way. The only difference is that the courier has different features.

Order encoder

Order encoder consists of courier encoder and route encoder. Encoded courier and route are concatenated and passed through a few layers MLP.

Route encoder

Let N be the maximum number of points in the route. In our system, N is a system parameter considered 2. Route encoder takes $n \leq N$ points, adds $N - n$ fake points (with zero coordinates), and then encodes all N points with point encoder. Then concatenate the result and pass through MLP.

Point encoder

The point encoder takes coordinates features as input and returns a vector. There are many ways how to make this transformation. I investigate this question in

Appendix.

Global encoder

This encoder works with global gamble features such as time and amounts of claims, couriers, and orders. Firstly, all these features are represented as a feature vector (for example, the weekday in one-hot-encoded), and secondly, a few layers of MLP are applied.

C Densified reward

Consider some metric represented as a function $MR(t) = \frac{\sum_{\tau=0}^{t} m_\tau}{\sum_{\tau=0}^{t} n_\tau}$, where n_t is an amount of new claims created in gamble G_t on step t and m_t is some statistics calculated on A_t or G_t.

For example, if m_t is the number of completed claims, we get the CR metric. If m_t is the number of assignments, then we get the assignment rate (AR) metric.

Let $M_t = \sum_{\tau=0}^{t} m_\tau$, $N_t = \sum_{\tau=0}^{t} n_\tau$ - cumulative amounts.
Then $MR(t) = \frac{\sum_{\tau=0}^{t} m_\tau}{\sum_{\tau=0}^{t} n_\tau} = \frac{M_t}{N_t}$.
Let's compute diffs of MR: $\Delta MR(t) = MR(t) - MR(t-1) = \frac{\sum_{\tau=0}^{t} m_\tau}{\sum_{\tau=0}^{t} n_\tau} - \frac{\sum_{\tau=0}^{t-1} m_\tau}{\sum_{\tau=0}^{t-1} n_\tau} = \frac{m_t N_{t-1} - n_t M_{t-1}}{(N_{t-1}+n_t)N_{t-1}}$

Thus, we can define MR densified reward:

$$R_{MR}(t) = \begin{cases} \frac{m_t N_{t-1} - n_t M_{t-1}}{(N_{t-1}+n_t)N_{t-1}}, & \text{if } N_{t-1} \neq 0 \\ \frac{m_t}{n_t}, & \text{if } N_{t-1} = 0 \text{ and } n_t \neq 0 \\ 0, & \text{else} \end{cases}$$

Considering $N_{-1} = 0$, it is easy to see that $MR(T) = \sum_{t=0}^{T} R_{MR}(t)$

That means that we may densify any MR reward without any approximations.

D Working with coordinates

Geo-coordinates are very specific features. That's why we need a special approach to work with them [19] [20]. I've tried 3 different approaches how to encode coordinates.

Linear encoder

This encoder takes coordinates, normalizes them, and applies a single linear layer.

Frequency encoder

These types of encoders are often used to work with geodata. Each coordinate is multiplied by a vector of either trainable parameters or not. Then, the sin and cos functions are applied to this vector.

Frequency encoder (united)

The same is true in the previous part, but instead of multiplying by different numbers, we just apply a linear layer: linear layer + sin/cos.

All 3 types of encoders could work either with trainable parameters or fixed ones. Firstly, I tried all these encoders on a simple supervised task of distance prediction between two random points. All encoders worked better in trainable mode, and a linear encoder worked slightly better than others.

On RL training (on easy data) results were the same. What is more, freezing pre-trained on distance-prediction task encoders does not help.

That's why, in all my experience, I used a linear encoder with trainable parameters.

References

1. Project github: https://github.com/limon8884/delivery-RL
2. Yang et al. 2022. Tackling the Crowdsourced Delivery Problem at Scale through a Set-Partitioning Formulation and Novel Decomposition Heuristic
3. Lei Y. M. et al. 2020. Dynamic workforce acquisition for crowdsourced last-mile delivery platforms
4. Ulmer M. W., Erera A., Savelsbergh M. 2022. Dynamic service area sizing in urban delivery
5. Saleh Z., Hanbali A. A., Baubaid A. 2024. Enhancing Courier Scheduling in Crowdsourced Last-Mile Delivery through Dynamic Shift Extensions: A Deep Reinforcement Learning Approach
6. Li J. et al. 2021. Heterogeneous Attentions for Solving Pickup and Delivery Problem via Deep Reinforcement Learning
7. Chen X., Ulmer M. W., Thomas B. W. 2022. Deep Q-learning for same-day delivery with vehicles and drones
8. Ahamed T. et al. 2022. Deep Reinforcement Learning for Crowdsourced Urban Delivery: System States Characterization, Heuristics-guided Action Choice, and Rule-Interposing Integration
9. Ding Y. et al. 2021. A City-Wide Crowdsourcing Delivery System with Reinforcement Learning
10. Behrendt A., Savelsbergh M., Wang H. 2023. A prescriptive machine learning method for courier scheduling on crowdsourced delivery platforms
11. Vaswani A. et al. 2017. Attention is all you need

12. Schulman J. et al. 2017. Proximal Policy Optimization Algorithms

13. Kuhn H. W. 1955. The Hungarian method for the assignment problem

14. Konda V., Tsitsiklis J. 1999. Actor-critic algorithms

15. Mataric M. J. 1994. Reward functions for accelerated learning

16. Luo Y. et al. 2020. Balance between efficient and effective learning: Dense2sparse reward shaping for robot manipulation with environment uncertainty

17. Bahdanau D., Cho K., Bengio Y. 2014. Neural machine translation by jointly learning to align and translate

18. Schulman J. et al. 2015 High-dimensional continuous control using generalized advantage estimation

19. Mai, G., et al. 2020. Multi-scale representation learning for spatial feature distributions using grid cells

20. Mai, G., et al. 2022. A Review of Location Encoding for GeoAI: Methods and Applications