Hybrid Genetic Search and Learn-to-Select for the EURO Meets NeurIPS 2022 Vehicle Routing Competition

Wenxuan Ao
Department of Electronic Engineering
Tsinghua University
Beijing, China
johnao73thu@gmail.com

Zefang Zong
Department of Electronic Engineering
Tsinghua University
Beijing, China
zongzefang@hotmail.com

Abstract

This document describes the team *kirchhoffslaw*'s submission to the EURO Meets NeurIPS 2022 Vehicle Routing Competition. We briefly introduce the competition and describe how we tune the parameters of the hybrid genetic search (HGS) solver for the static problem and the design of a reinforcement learning model to select which order to dispatch in the dynamic problem. The source code is released on `https://github.com/tsinghua-fib-lab/learn-to-select`.

1 Introduction

In the EURO Meets NeurIPS 2022 Vehicle Routing Competition [1], there are static and dynamic versions of the vehicle routing problem with time windows (VRPTW). For the static version, all the information of customers are known at the beginning, including the demand, position, time window, and the asymmetric duration matrix for commuting between the depot and customers. The capacity of the vehicles is limited but the number of used vehicle is unlimited. It is guaranteed that sending one vehicle for each customer is a feasible solution. For the dynamic version, the orders of customers arrive hourly (by *epoch*), and we can choose not to dispatch all the orders in the hope that they can be combined with future orders, saving the cost. The order is called *must-dispatch* if postpone it to next epoch will violate the time window constraint. The dynamic orders are generated by sampling from a static problem, whose definition is given in the beginning and can be used as prior information.

In the following, we will describe our method for the static and dynamic problems in detail.

2 Method for Static Problem

HGSTW, the baseline static problem solver provided by the organizers, has a great performance out of the box. Therefore, we use HEBO[1], the winner of the NeurIPS 2020 Black-Box Optimisation Challenge, to tune the parameters of the solver.

First, We divide by size the 250 test cases into type A (≤ 300), B ($> 300, \leq 500$) and C (> 500), since the allowed solving time is different for the 3 types. Then, we tune the parameters listed in Table 1 for each type. Note that the first 3 parameters takes the same value.

The optimum parameters are shown in Table 2, resulting from $1000+$ trials for each row.

[1] https://github.com/huawei-noah/HEBO

EURO Meets NeurIPS 2022 Vehicle Routing Competition Workshop.

Commandline Parameter Name	Tuning Variable Name	Meaning
fractionGeneratedFurthest	fGNFS	proportion of individuals constructed by nearest-first
fractionGeneratedNearest	fGNFS	proportion of individuals constructed by nearest-first
fractionGeneratedSweep	fGNFS	proportion of individuals constructed by sweep
generationSize	incP	generation size
initialTimeWarpPenalty	iTWP	penalty for time window violation
intensificationProbabilityLS	iPLS	probability of doing intensification moves
maxToleratedCapacityViolation	mTCV	maximum tolerance for capacity violation
maxToleratedTimeWarp	mTTW	maximum tolerance for time window violation
minCircleSectorSizeDegrees	mCSSD	minimum size for circle sectors in degrees
minimumPopulationSize	minP	minimum population size
nbClose	nbC	number of closest individuals considered when calculating diversity contribution
nbElite	nbE	number of elite individuals considered when calculating diversity contribution
nbGranular	nbG	limit on the number of moves in the RI local search
repairProbability	rP	probability of repairing an individual if it is infeasible after local search
targetFeasible	tF	target feasible solution ratio

Table 1: The name and meaning of the tuned parameters

Type	Time limit	seed	nbG	mTCV	nbC	mTTW	iTWP	tF	rP	fGNFS	mCSSD	iPLS	nbE	minP	incP
A	1	1	35	30	5	420	1	0.2	50	0.05	12	11	5	35	20
A	2	1	30	15	9	360	1	0.3	30	0.1	19	20	5	50	20
A	3	1	35	50	3	180	1	0.2	50	0.05	11	10	6	45	30
A	5	1	35	30	6	180	1	0.2	70	0.2	16	12	6	45	20
B	5	1	35	15	1	540	1	0.2	50	0.05	20	11	5	50	35
B	10	1	35	30	5	60	1	0.2	50	0.1	12	11	5	45	35
C	8	43	30	25	5	480	1	0.2	50	0.05	16	10	3	25	50
C	15	1	70	10	10	180	1	0.15	60	0.25	15	17	8	35	40

Table 2: Optimum parameters for different problem types and time limits (/min)

3 Method for Dynamic Problem

On the high level, we need to first decide whether to deal with each order in every epoch, then plan the routes for the selected orders. The second step can be done using the solver in the static problem. And the decisions in the first step are made by heuristics or an RL agent.

In this section, we will briefly go over some heuristics we have tried, and describe the final RL method in detail.

3.1 Heuristics

The basic intuition is that, if we dispatch a vehicle solely for one faraway customer, the cost is too high. And we should wait long enough for more customers to appear near it, so that the average cost is acceptable.

To this end, we try to divide the customers into K regions by fitting a Gaussian mixture model (GMM) on the coordinates of the customers based on the associated static problem, as shown in Figure 1. If there are any must-dispatch customers in a region, all customers in the region will be dispatched. Otherwise, all will be postponed. The feature used for GMM can also incorporate other information, like the angle and distance to the depot.

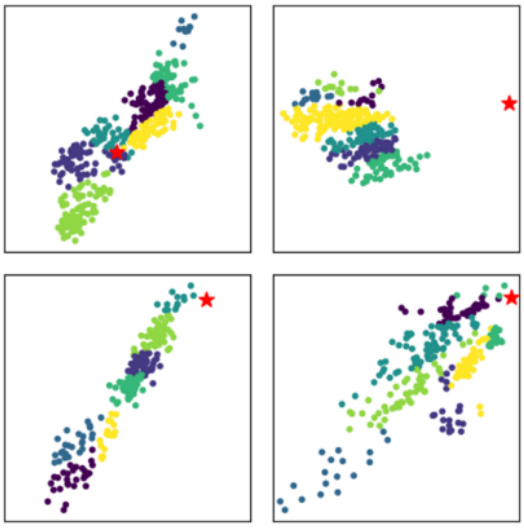

Figure 1: **Four examples of the division of the customers.** Customers are represented by colored dots and depot is represented by red star.

Another simple but effective heuristic is to first solve the routes for all available customers, and then postpone all the customers that have no must-dispatch orders in the same route. Optionally, we can solve again for the remaining customers and repeat this process until all routes have at least one must-dispatch customers in it.

3.2 Learn to select with RL

Following the recent trend of *Learn to Optimize* [2, 3, 4], we try to learn a better heuristic for selecting which order to dispatch using reinforcement learning.

In the MDP formulation, it is natural to take each epoch as a step. And to make it Markovian, a GRU module is added to gather historical information. The state consists of the following:

- global info
  - current epoch number
  - number of remaining epochs
  - number of must-dispatch orders
  - total distance of all must-dispatch orders
  - total distance of all non-must-dispatch orders
- features of all of the available orders in current time step
  - positions
  - time windows
  - number of nearby orders
  - distance to nearby orders, etc.

The action is whether to dispatch each order. And the reward is given at the last epoch, which is the difference in the total cost of the routes compared to that of the GREEDY baseline. Here we use relative cost (subtract the cost of GREEDY baseline from the total cost) to reduce the estimation variance.

The overall architecture is shown in Figure 2. To calculate the state representation, the features of the orders are first encoded into 64-dimentional vectors by a MLP module, then pooled using max pooling and average pooling, concatenated with the global info features, encoded with another MLP module, and finally go through the GRU module. The critic module is an MLP that takes the state representation as input and outputs state value estimation. The action module is an MLP that for

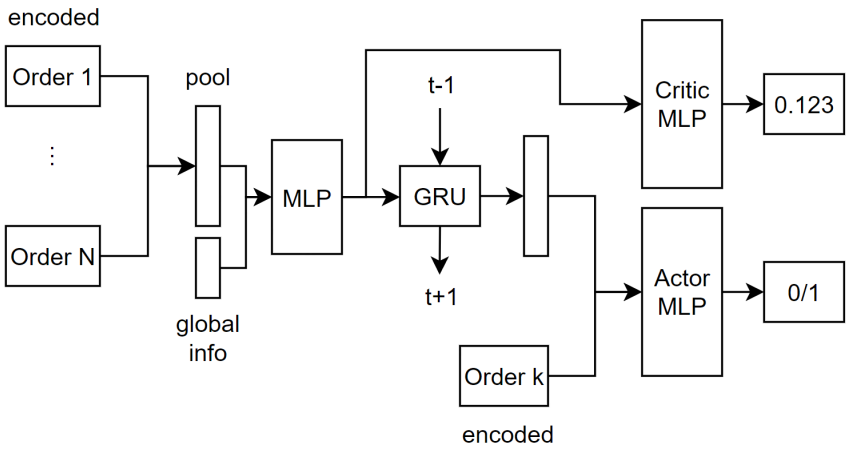

Figure 2: The RL model architecture.

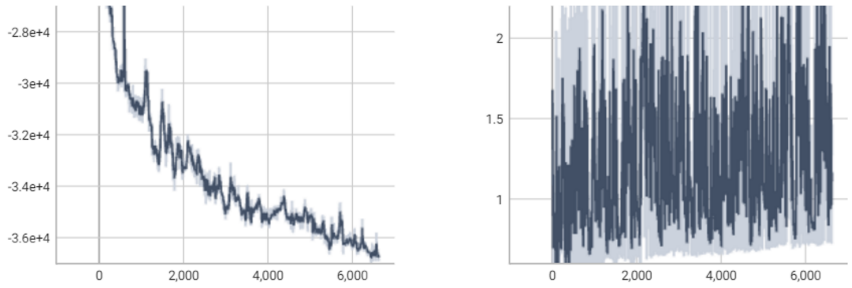

Figure 3: The relative cost (compared with GREEDY) and reward during training.

each order takes the state representation and the encoded vector of that order as input and outputs the probability of whether to dispatch this order or not.

For training, we use 60 problems from the given dataset and sample from the environments in parallel. To increase sampling speed, we limit the solving time of HGS for each step to no longer than 10s. Overall, we get a training speed of roughly 700 steps per day. As Figure 3 shows, the relative cost is steadily decreasing up to the moment we stop. It is a pity that the training speed is pretty slow and we could have obtained a better performance given more time.

References

[1] W. Kool, L. Bliek, D. Numeroso, R. Reijnen, R. R. Afshar, Y. Zhang, T. Catshoek, K. Tierney, E. Uchoa, T. Vidal, and J. Gromicho, "The EURO meets NeurIPS 2022 vehicle routing competition," 2022.

[2] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada* (C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, eds.), pp. 2692–2700, 2015.

[3] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, OpenReview.net, 2019.

[4] Z. Zong, M. Zheng, Y. Li, and D. Jin, "MAPDP: cooperative multi-agent reinforcement learning to solve pickup and delivery problems," in *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pp. 9980–9988, AAAI Press, 2022.