Winning Approach for the EURO-NeurIPS 2022 Dynamic Vehicle Routing Competition

Léo Baty[1], Kai Jungel[2], Patrick Klein[2] Axel Parmentier[1], Maximilian Schiffer[2]

[1] CERMICS, École des Ponts, France
{leo.baty, axel.parmentier}@enpc.fr
[2] Technical University of Munich (TUM), Germany
{kai.jungel, patrick.sean.klein, schiffer}@tum.de

Mots-clés : *combinatorial optimization, machine learning, vehicle routing, multi-stage optimization, hybrid genetic search, EURO-NeurIPS challenge*

1 Introduction

The *EURO meets NeurIPS 2022 Vehicle Routing Competition*[1] focuses on the usual static *Capacitated Vehicle Routing Problem with Time Windows* (VRPTW), as well as a dynamic variant which is the focus of this paper.

The objective of the dynamic VRPTW is to build routes for a fleet of capacitated vehicles, in order to serve all customer requests within their given time windows. The objective is to minimize the total travel time. However, requests are not known in advance: they arrive continuously during the day. Every hour, the decision maker chooses which requests to dispatch and builds routes to serve them. Each request must be served before the end of its time window. A request whose time window allows serving it during the next hour can be postponed. Once a route has been built, it cannot be modified.

The static VRPTW has been extensively studied in the literature, including exact approaches [5], as well as heuristic ones, among which the hybrid genetic search [6, 4]. This can be seen in the results of the challenge: the objective of the algorithm proposed by the 5th team on the static problem is 0.02% worse than the best solution found. The question of building an efficient heuristic for the dynamic problem remains far more open: the policy proposed by the 5th team is 4% worse than the best proposed.

Our main contribution is a policy for the dynamic VRPTW. It relies on a Deep Learning pipeline with a prize collecting VRPTW combinatorial optimization layer. It ranked first of the EURO-NeurIPS competition, both the on dynamic problem ranking and the global ranking. This pipeline requires a subroutine for solving the prize collecting VRPTW, for which we introduce the prize collecting hybrid genetic search, a variant of the hybrid genetic search [6] adapted for the prize collecting VRPTW. As a side contribution, we implemented an extension of the `InferOpt.jl` Julia library [3] with a generic support for generalized combinatorial optimization oracles, and in particular for training our pipeline.

In Section 2, we introduce the dynamic VRPTW and the notations. Section 3 details each component of our deep learning pipeline, while Section 4 explains the learning approach used. Finally, Section 5 presents our team results at the end of the challenge.

[1]See https://euro-neurips-vrp-2022.challenges.ortec.com/

2 Dynamic VRPTW: problem statement

Definitions Let $[T]$ be the discrete *time horizon* divided into 1-hour *epochs.* A *request* contains four components: coordinates p, a time window $[\ell, u]$, a demand value q, and a service time s. Travel time between locations p and p' is denoted by $d_{p,p'}$. A *route* is a sequence of requests starting from and ending at the *depot* (denoted by p_0). When a feasible route is built, each request of the route is said to have been *served.*

Dynamic problem The dynamic VRPTW can be modeled as a Markov Decision Process. The *state* x_t of the system at a given epoch t is entirely defined by its set of requests, composed of the requests arrived at epoch t, and those arrived before but not yet served.

At each epoch t, the decision maker chooses an *action*: he decides to build routes to serve some requests from x_t. A route built in epoch t cannot start before $t+1$. A given request $(p, [\ell, u], q, s)$ is *feasible* at epoch t if we can build a route to serve it in time, i.e. if we can reach it from the depot starting at $t+1$:

$$d_{p_0,p} + t + 1 \leq u. \tag{1}$$

Some requests from x_t must be served during this epoch, otherwise they become unfeasible in the next epoch and violate (1). We denote by $M_t \subset x_t$ this set of *must-dispatch* requests. Let $\mathcal{R}(x_t)$ be the set of feasible routes for state x_t (i.e. following demand and time window constraints), and d_r the total travel time duration of a feasible route r. For each epoch $t \in [T]$, we define decision variables $y_{t,r} \in \{0,1\}$, equal to 1 if and only if the route $r \in \mathcal{R}(x_t)$ is chosen at t. Each request can be served at most once, except for *must-dispatch* ones which need to be fulfilled exactly once. The set of feasible decisions is denoted $\mathcal{Y}(x_t)$. This leads to the following constraints on these decision variables:

$$\sum_{r \ni v} y_{t,r} \leq 1, \; \forall t \in [T], \; v \in x_t \backslash M_t, \tag{2a}$$

$$\sum_{r \ni v} y_{t,r} = 1, \; \forall t \in [T], \; v \in M_t. \tag{2b}$$

When taking decision y_t at time t in state x_t, a cost $\sum_{r \in \mathcal{R}(x_t)} d_r y_{t,r}$ is incurred. Additionally, all served requests (denoted Y_t) are removed, and, new requests A_{t+1} arrive at the start of the next epoch $t+1$.

$$\begin{aligned} x_{t+1} &= A_{t+1} \cup x_t \backslash Y_t, \; \forall t \in [T-1], \\ x_1 &= A_1. \end{aligned} \tag{3}$$

A *policy* π is defined as a mapping from states x_t to probabilities $\pi(y|x_t)$ of selecting each given decision $y \in \mathcal{Y}(x_t)$ in the current state x_t:

$$y_t \sim \pi(\cdot|x_t), \; \forall t \in [T]. \tag{4}$$

Therefore, the dynamic VRPTW can be formulated as

$$\min_{\pi \in \Pi} \quad \mathbb{E}_\pi \left[\sum_{t \in [T]} \sum_{r \in \mathcal{R}(x_t)} d_r y_{t,r} \right] \tag{5}$$

with Π the set of all possible policies.

A *deterministic policy* π^{d} is defined as a mapping from states x_t to decisions $y_t \in \mathcal{Y}(x_t)$.

Challenge instances In the challenge, newly arrived requests A_t are drawn from an associated static VRPTW instance with N requests $(p_i, [\ell_j, u_j], q_k, s_l)_{i \in [N]}$. At each epoch t, 100 requests are uniformly drawn from $\{(p_i, [\ell_j, u_j], q_k, s_l) \mid (i,j,k,l) \in [\![1, N]\!]^4\}$, and only feasible requests following (1) are kept. The associated static instance (and therefore the full request probability distribution) is known at the start of the time horizon and can be used by the policy.

3 Our policy based on a Deep Learning pipeline

Pipeline The set of feasible decisions $\mathcal{Y}(x_t)$ corresponds to the feasible set of a prize collecting (static) VRPTW. Indeed, in a prize collecting VRPTW, the decision maker selects requests v to serve from a set of available requests x_t in order to maximize its profit, which is the difference between the prizes θ_v collected for serving selected requests v and the driving costs d_{uv} incurred when driving between requests u and v along built routes.

We therefore propose a policy that solves a prize collecting VRPTW at each epoch to make its decision. This requires to build a complete prize collecting VRPTW instance from the epoch state x_t defined in previous Section 2. Unfortunately, while x_t is naturally defined as the set of requests arrived but still not dispatched before t, and $(d_{uv})_{(u,v)\in x_t^2}$ as the travel cost matrix between requests, there is no natural definition of the request prizes θ_v. For this, we use a neural network φ_w to compute $(\theta_v)_{v\in x_t}$ given the epoch state x_t. Our deterministic policy π_w^{d} which maps a state x_t to a decision $y_t \in \mathcal{Y}(x_t)$ consists in the following pipeline:

$$\begin{tikzcd} {} \arrow[r, "\text{State}", "x_t"'] & \boxed{\begin{matrix}\text{Neural Network}\\ \varphi_w\end{matrix}} \arrow[r, "\text{Requests prizes}", "\theta_v,\, \forall v\in x_t"'] & \boxed{\boxed{\begin{matrix}\text{Prize Collecting}\\ \text{VRPTW } f\end{matrix}}} \arrow[r, "\text{Epoch routes}", "y_t"'] & {} \end{tikzcd} \tag{6a}$$

$$\pi_w^{\mathrm{d}}\colon x_t \longmapsto y_t = f(\varphi_w(x_t)) \tag{6b}$$

Our policy is therefore parametrized by the network weights w.

Two challenges must still be addressed. First, we must propose an algorithm f to solve the prize collecting VRPTW. For this purpose, we adapt the hybrid genetic search of [6] to that problem. Second, the policy being parametrized by the weights w of the neural network, we must therefore find weights w that lead to a practically efficient policy. The learning approach is detailed in Section 4.

Prize Collecting Hybrid Genetic Search From now on, instead of considering the set partitioning formulation (5), we use a compact formulation and embed decisions $\mathcal{Y}(x_t)$ to $\{0,1\}^{x_t^2}$. Binary decision variables $(y_{u,v})_{(u,v)\in x_t^2}$ are equal to 1 if a chosen route chains requests u and v, and 0 otherwise. The prize collecting VRPTW, can be written as

$$\max_{y\in\mathcal{Y}(x_t)} \sum_{(u,v)\in x_t^2} (\theta_v - d_{u,v})y_{u,v}. \tag{7}$$

The combinatorial optimization oracle f introduced in (6) is therefore defined as

$$f\colon \theta \longmapsto \operatorname*{argmax}_{y\in\mathcal{Y}(x_t)} \theta^\top g(y) + h(y), \tag{8}$$

with $g(y) = (\sum_{u\in x_t} y_{u,v})_{v\in x_t}$, and $h(y) = -\sum_{(u,v)\in x_t^2} d_{u,v}y_{u,v}$. Note that h also depends on the cost matrix, which is omitted for notation simplicity. $g(y)$ can be interpreted as the dispatch decision: each of its components v is equal to 1 if and only if the request v is served. $-h(y)$ is the incurred travel cost.

In order to solve this problem, we use a prize collecting hybrid genetic search, a variant of the state-of-the-art hybrid genetic search [6] algorithm for the static VRPTW. A usual hybrid genetic search is a genetic algorithm. It maintains a population of solutions at each iteration, which evolves with crossover and mutation operations as in usual genetic algorithms, but are also combined with a neighborhood search heuristic.

First, our algorithm extends the solution encoding in order to account for optional customers: we add a *request set* to the encoding, which determines the set of customers served by the solution. Second, from the two crossover operators, we remove one because experiments showed that it did not contribute to the algorithm's performances for our instances. We also modify the second crossover operator such that is preserves the request set of the first parent. Finally, we introduce two new mutation operators that modify the request set. The first operator either removes some served customers from the request set, or inserts some currently unserved customers into the request set. The second operator optimizes the request set of a given solution: it first removes any customer causing incurring a cost higher than the customer's profit, and then reinsert any profitable customer which is not part of the current solution. The operator perturbs insertion and removal costs with a random factor. We apply the second operator only after the neighborhood search converges, in order to avoid removing an excessive amount of customers due to poor solution quality.

4 Learning approach

In order to achieve good performances with our policy (6), we need to find weights w of the neural network φ_w, such that output routes y_t of our pipeline are a good policy for any input instance. Our learning approach is to train our pipeline to imitate the decision taken by an anticipative policy.

Anticipative policy imitated and training set generation On a given instance, we can draw a full scenario (all the arriving request at each epoch). We then solve the full instance using a hybrid genetic search. In order to obtain a feasible solution of the dynamic problem for the drawn scenario, we modify time windows such that requests arriving at a given epoch cannot be served by a route starting before it. We rebuild the decisions that would have been taken at each epoch an obtain a solution. Note that this corresponds to an anticipative policy which cannot be used in practice because it needs to know in advance all arriving requests. However, we can use it to build a dataset of instances labeled with route decisions we want to imitate. We denote this dataset as

$$\mathcal{D} = \{(x_{t_1}^1, \overline{y}_{t_1}^1), \dots, (x_{t_n}^n, \overline{y}_{t_n}^n)\}. \tag{9}$$

Learning problem Usually, in classical supervised learning, the learning problem is formulated by defining a loss function $\mathcal{L}$ that measures output quality, and finds $w^\star$ minimizing it on the known training data $\mathcal{D}$:

$$w^\star = \operatorname*{argmin}_w \sum_{i=1}^n \mathcal{L}(\varphi_w(x_{t_i}^i), y_{t_i}^i). \tag{10}$$

The optimization problem (10) is usually solved using a gradient algorithm that relies on automatic differentiation to back-propagate gradients through the pipeline.

Loss function In order to imitate target solutions $(\overline{y}_{t_i}^i)_{i\in[n]}$ of our dataset, we want them to be as close as possible to the optimal solutions of (8). A natural loss function is therefore the non-optimality of the target epoch routes $\overline{y}_t$ respect to the combinatorial optimization oracle:

$$\mathcal{L}(\theta, \overline{y}_t) = \max_{y\in\mathcal{Y}(x_t)} \{\theta^\top g(y) + h(y)\} - (\theta^\top g(\overline{y}_t) + h(\overline{y}_t)) \tag{11}$$

Unfortunately, $\mathcal{L}$ is not smooth and $\theta = 0$ is an optimal solution, which leads to difficult gradient optimization. This is due to the combinatorial layer f being piecewise constant.

A regularization approach has been recently introduced in the literature [1] to smooth linear optimization oracles of the form $\theta \mapsto \operatorname{argmax}_y \theta^\top y$. It consists in perturbing θ with an additive noise. We extend this setting to oracles which are affine in θ but not necessarily linear in y, which is exactly the form of (8). The perturbed combinatorial optimization oracle is defined by

$$\widehat{f}_\varepsilon \colon \theta \longmapsto \mathbb{E}\left[\operatorname*{argmax}_{y\in\mathcal{Y}(x_t)} (\theta + \varepsilon Z)^\top g(y) + h(y)\right] = \mathbb{E}[f(\theta + \varepsilon Z)], \tag{12}$$

with $Z \sim \mathcal{N}(0,1)$ an additive Gaussian perturbation, and $\varepsilon \in \mathbb{R}_+$ a fixed hyperparameter. In practice, the expectation in (12) is intractable, but can be approximated by Monte-Carlo sampling. The associated regularized loss is defined as follows:

$$\mathcal{L}_\varepsilon^{FY}(\theta, \overline{y}_t) = \mathbb{E}\left[\max_{y\in\mathcal{Y}(x_t)} (\theta + \varepsilon Z)^\top g(y) + h(y)\right] - (\theta^\top g(\overline{y}_t) + h(\overline{y}_t)). \tag{13}$$

This loss is differentiable, and admits the following subgradient

$$g(\widehat{f}_\varepsilon(\theta)) - g(\overline{y}_t) \in \partial_\theta \mathcal{L}_\varepsilon^{\text{FY}}(\theta, \overline{y}_t). \tag{14}$$

This loss can be shown to have nice properties using Fenchel Duality, which is the reason why it is called a Fenchel-Young loss [2]. We denote by Ω_ε the Fenchel conjugate of $\theta \mapsto \mathbb{E}\left[\max_{y\in\mathcal{Y}(x_t)}(\theta + \varepsilon Z)^\top g(y) + h(y)\right]$. We have that $\theta \mapsto \mathcal{L}_\varepsilon^{\text{FY}}(\theta, \overline{y}_t) + \Omega_\varepsilon(\overline{y}_t)$ is positive by Fenchel's inequality, convex, and the minimum 0 is reached at θ if and only if $g(\widehat{f}_\varepsilon(\theta)) = g(\overline{y}_t)$.

Support for *generalized maximizer* oracles of the form $\theta \mapsto \operatorname{argmax}_y \theta^\top g(y) + h(y)$ has been implemented in the open source Julia library `InferOpt.jl` [3]. The implementation is generic and not only specific to the prize collecting VRPTW. Note that for the challenge, we used the Julia implementation for prototyping, and a specific Python implementation for hyperparameter optimization and easier compatibility with the challenge's code environment.

Inexact oracle In this section, we did not mention that the prize collecting hybrid genetic search is an inexact combinatorial oracle: it's a (meta)heuristic algorithm which does not necessarily output an optimal solution, but only a "good" feasible solution of the prize collecting VRPTW (8). However, this is not a problem in practice: [3] shows that the computed gradient with an inexact oracle is a good approximation of the real one when the inexact oracle outputs solutions close enough to optimal ones.

5 Results

Dataset On average, a problem duration is around 6 epochs. Our dataset consists in all epochs from 20 dynamic instances, that is 110 training samples. Label target solutions $\overline{y}_t$ are built by drawing only 60 new requests at the start of each epoch instead of 100 in the challenge evaluation. These smaller instances enable the prize collecting hybrid genetic search to output solutions closer to the optimal ones, which helps the learning process. We observe that our pipeline trained on smaller instances generalizes quite well on larger ones.

Neural Network architecture and hyperparameters We use a classical multi-layer perceptron as our neural network architecture. It's a small network composed of 4 layers with output size 10 each, and a final layer with output size 1. We use ReLU as activation function, and the *Adam* optimizer with 0.01 learning rate. We use $\varepsilon = 1$, and 20 samples for the Monte Carlo evaluation of the expectation in (12). Finally, the runtime for the prize collecting hybrid genetic search during training is set to 120 seconds, in order to ensure convergence of the heuristic. We selected the weights w from the training epoch 61, which was the one performing best when evaluating on a separate validation dataset.

Results With our team `Kléopatra`, we ended up at first place on the final leaderboard, including 2nd place on the static VRPTW, and 1st place on the dynamic VRPTW (see Table 1).

Rank	Team name	Dynamic cost	Static rank	Dynamic rank
1	Kléopatra	348831.56	2	1
2	OptiML	359270.09	1	3
4	Team_SB	358161.36	3	2
3	HustSmart	361803.57	5	4
5	Miles To Go Before We Sleep	369098.13	4	7

TAB. 1: Top 5 teams of the final leaderboard

For a more detailed benchmark, we evaluate our policy on 2252 instance-seed combinations, against the 5 baseline strategies given by the challenge organizers, as well as the anticipative policy. For each combination and strategy, we evaluate the gap to the best solution found of all strategies, and display a box plot of this metric in Figure 1 below. Our learned policy performs better than all five baseline strategies, with an average gap of 4.4% respect to the best solution found, and is relatively close to the anticipative policy.

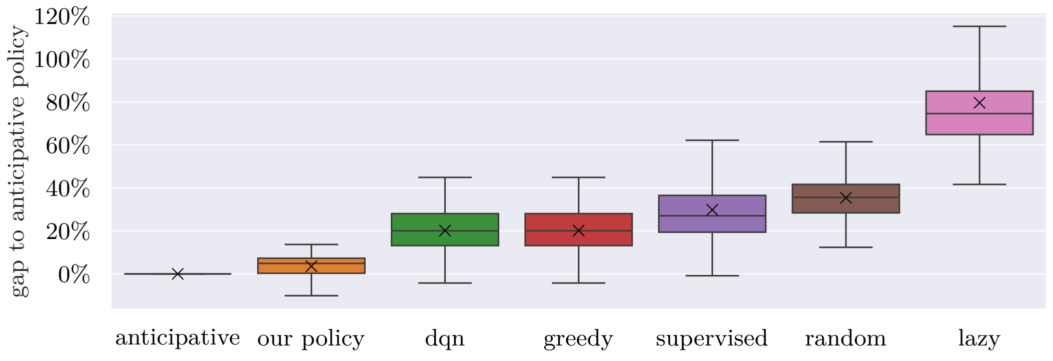

FIG. 1: Box plot comparing the gap in % to the anticipative policy. Crosses give mean values.

References

[1] Quentin Berthet et al. "Learning with Differentiable Perturbed Optimizers". en. In: *arXiv* (June 2020). URL: http://arxiv.org/abs/2002.08676 (visited on 02/01/2022).

[2] Mathieu Blondel, André FT Martins, and Vlad Niculae. "Learning with Fenchel-Young losses." In: *J. Mach. Learn. Res.* 21.35 (2020), pp. 1–69.

[3] Guillaume Dalle et al. *Learning with Combinatorial Optimization Layers: a Probabilistic Approach*. arXiv:2207.13513 [cs, math, stat]. July 2022. DOI: 10.48550/arXiv.2207.13513. URL: http://arxiv.org/abs/2207.13513 (visited on 10/07/2022).

[4] Wouter Kool et al. "Hybrid Genetic Search for the Vehicle Routing Problem with Time Windows: a High-Performance Implementation". In: (2022).

[5] Artur Pessoa et al. "A generic exact solver for vehicle routing and related problems". en. In: *Mathematical Programming* 183.1 (Sept. 2020), pp. 483–523. ISSN: 1436-4646. DOI: 10.1007/s10107-020-01523-z. URL: https://doi.org/10.1007/s10107-020-01523-z (visited on 11/14/2022).

[6] Thibaut Vidal. *Hybrid Genetic Search for the CVRP: Open-Source Implementation and SWAP* Neighborhood*. arXiv:2012.10384 [cs]. Oct. 2021. DOI: 10.48550/arXiv.2012.10384. URL: http://arxiv.org/abs/2012.10384 (visited on 09/30/2022).