Neural Order-First Split-Second Algorithm for the Capacitated Vehicle Routing Problem

Ali Yaddaden, Sébastien Harispe, Michel Vasquez

Neural Order-First Split-Second Algorithm for the Capacitated Vehicle Routing Problem

Ali Yaddaden[1], Sébastien Harispe[1⋆], and Michel Vasquez[1]

EuroMov Digital Health in Motion, Univ Montpellier, IMT Mines Ales, Ales, France
{firstname.lastname}@mines-ales.fr

Abstract. Modern machine learning, including deep learning models and reinforcement learning techniques, have proven effective for solving difficult combinatorial optimization problems without relying on handcrafted heuristics. In this work, we present NOFSS, a Neural Order-First Split-Second deep reinforcement learning approach for the Capacity Constrained Vehicle Routing Problem (CVRP). NOFSS consists of a hybridization between a deep neural network model and a dynamic programming shortest path algorithm (Split). Our results, based on intensive experiments with several neural network model architectures, show that such a two-step hybridization enables learning of implicit algorithms (i.e. policies) producing competitive solutions for the CVRP.

Keywords. Neural Combinatorial Optimization Capacitated Vehicle Routing Problem Order-first Split-second Deep Reinforcement Learning

1 Introduction

Modern machine learning, including deep learning models and reinforcement learning techniques, have proven effective for solving difficult combinatorial optimization problems without relying on handcrafted heuristics [1]. The framework known as Neural Combinatorial Optimization (NCO), which proposes to solve combinatorial optimization problems using recent neural networks architectures, is in this context widely studied for routing problems such as the traveling salesman problem (TSP) [2–5] and the capacitated vehicle routing problem (CVRP) [6, 5].

Current NCO approaches implement a construction-based strategy. For the CVRP, such approaches build (i.e. construct) candidate solutions step by step, by selecting at each time step either to visit a client or to go back to the depot to refill, until each client is served. The action to perform at each construction step is chosen based on a probability distribution that will be estimated by a deep neural network, either using supervised or reinforcement learning. This discrete probability distribution defines the probability that an extension of the partial solution under construction, considering each available choices (unsatisfied clients and depot), will lead to the optimal solution. Considering such construction-based NCO approaches, solving the CVRP is therefore reframed as a learning goal aiming to obtain a good estimate of the probability distribution, such as step decisions based on this estimate minimize solution costs.

Using such an approach, the models handle both clients routing and returns to depot. In this context, choices of when to return to the depot are critical. Indeed, more returns to the depot can *de facto* lead to candidate solutions with a number of tours[1] greater than the optimal one. This will result in models failing to efficiently learn interesting resolution strategies, i.e. routing *policies*, due to poor quality candidate solutions, and/or large computational costs inducing prohibitive learning process (millions of learning steps). Handcrafted heuristics and metaheuristics may nevertheless be used to handle return to depot by using an exact tour splitting algorithm - solving a shortest path problem in an auxiliary graph that represents the clients' visit order [7, 8]. Inspired by this problem decomposition, this paper presents NOFSS, Neural Order-First Split-Second, a novel two-step learning-based approach proposing to:

1. Learn how to order clients into a giant tour, using a deep neural network.
2. Optimally split the giant tour into a feasible solution using an exact split algorithm.

⋆ This work used HPC resources of IDRIS (allocation 2022-AD011011309R2) made by GENCI.

[1] A tour is the ordering of clients the vehicle will visit before returning back to the depot. The optimal number of tours will therefore depend on client's demands and vehicle capacity.

NOFSS is a generic approach that will be introduced and tested in the context of CVRP, even if it may be used for a larger class of routing problems. NOFSS relies on a deep neural network that learns a giant tour policy and a dynamic programming algorithm, called Split [8]. Split modifies the giant tour into a feasible solution with respect to vehicle capacity and clients demands. It acts as an oracle that provides feedback on the quality (the total travelled distance) of the giant tour generated from our neural network. This makes it possible to train the NOFSS model through REINFORCE algorithm.

Alongside NOFSS introduction, we present an extensive comparison of various NOFSS and NCO models with state-of-the-art CVRP (meta)heuristics. Results show that, by exploring the search space of giant tours, NOFSS allows to implicitly learn competitive routing policies. [2]

The paper is organized as follows: Section 2 formally introduces the CVRP and notations; Section 3 introduces related work focusing on approaches based on machine learning; Section 4 presents NOFSS; Section 5 presents the experimental protocol as well as results. Discussions and perspectives conclude the paper.

2 Problem Statement

The Capacitated Vehicle Routing Problem (CVRP) is one of the basic types of routing problems where information associated with the clients, the depot and the vehicles are deterministic and known in advance. We consider a set of n clients dispatched on the Euclidean plan and a single depot. In the depot, there is a fleet of homogeneous vehicles with identical transport capacity C. We associate to the clients their coordinates (x_i, y_i) and their demands of goods to deliver $0 \leq d_i \leq C$ $(i \in \{1, ..., n\})$. We associate to the depot its coordinates (x_0, y_0). The demands cannot be split, meaning that a vehicle must satisfy the demand at once. The objective is to minimize the total travelled distance when serving all the clients.

The problem can also be formulated using graph theory [9]. We consider a complete graph $G(V, E)$, where $V = \{0, ..., n\}$ is the vertex set (the vertex 0 represents the depot) and $E = \{(u, v) \in V \times V, u \neq v\}$ is the edge set. We associate with each edge a cost defined as the distance between two vertices. We can represent it as a cost matrix D where $D_{uv} = \sqrt{(x_u - x_v)^2 + (y_u - y_v)^2}$, $(u, v) \in E$. The goal is in this case to find simple circuits called tours such that all clients are served without transgressing the vehicles' capacity and the total travelled distance is as minimum as possible.

3 Related Work

3.1 Neural Combinatorial Optimization (NCO) for the CVRP

We refer to the use of end-to-end deep neural network approaches for solving difficult combinatorial optimization as the Neural Combinatorial Optimization (NCO) framework [3]. In this section, we review the use of this framework to learn construction-based policies for routing problems.

Although the use of neural networks for solving combinatorial optimization problems dates back longer than the appearance of modern deep learning architectures [10], their use has faded away in favor of more efficient metaheuristics. The success of deep learning and reinforcement learning has revived the interest in studying deep neural networks for solving this class of problems. More precisely, with the appearance of the sequence-to-sequence type approaches and the attention mechanism. The general framework (Figure 1) considers two neural networks called respectively encoder and decoder, which can be of different types. The encoder generates the *embeddings* of each element of a problem instance (clients and depot). Embeddings can be viewed as an alternative representation of the element in a higher dimension vector space ($\mathbb{R}^d$ with generally $d = 64$ or $d = 128$). This representation is intended to encompass meaningful features that will be used during the decoding phase. The decoder uses the history of the already visited elements (clients or depot) to compute a query vector that summarizes the solution under construction through a single vector. The query along with the embeddings are used to compute a probability distribution of selecting

[2] Our implementation and results will be available on the following repository https://github.com/AYaddaden/NOFSS

the next element via an attention module. To do so, the attention module confronts the query $q \in \mathbb{R}^d$ to the elements embeddings $e_i \in \mathbb{R}^d$ in order to give attention scores s_i either via a scaled dot-product (i.e. $s_i = \frac{q \cdot e_i^T}{\sqrt{d}}$) or via an additive attention defined as $s_i = v^T \cdot tanh(W_q \cdot q + W_e \cdot e_i)$ with $W_q, W_e \in \mathbb{R}^{d \times d}$, $v \in \mathbb{R}^d$ being learnable parameters. The scores s_i will be converted into a probability distribution by a softmax function[3].

Pointer Networks [2] was the seminal work that considered training LSTM-based encoder and decoder along with an additive attention module via supervised learning on a dataset of TSP instances. The approach successfully solved instances of sizes between 10 and 50 cities. It was next improved by using a policy-based reinforcement learning algorithm for training, namely REINFORCE with critic baseline, thus avoiding the need of a supervision, i.e. to have ground truth optimal solutions for the TSP dataset's instances [3]. Reinforcement learning proved to be more effective for training models on instances of size between 20 and 100 cities, thus achieving better results than Pointer Networks.

Nazari et al. [6] applied the NCO approach to CVRP. Their model considered 1D convolutions instead of an LSTM encoder in order not to bias the model on the inputs' order – LSTM are indeed better suited for modeling sequences where input's order matters. Comparison with classic CVRP algorithms (Clarke and Wright *savings* heuristic and the Sweep algorithm) shows that the deep neural network model performs better on training and test instances' sizes ranging from 10 to 100 clients. It appears also, that the choices of the encoder and decoder are of extreme importance in order to improve the learned policy. The Attention Model (AM) improves the results on the TSP and the CVRP by introducing a model entirely based on the attention mechanism [5]. It uses a Transformer encoder and computes the query vector using a Multi-head attention [11]. The Transformer encoder allows taking into account the graph structure of the TSP and the CVRP in the same way Graph Neural Networks do, thus giving a better representation of the instances. Also, they introduce a new baseline for the REINFORCE algorithm; a greedy rollout baseline that is a copy of AM that gets updated less often.

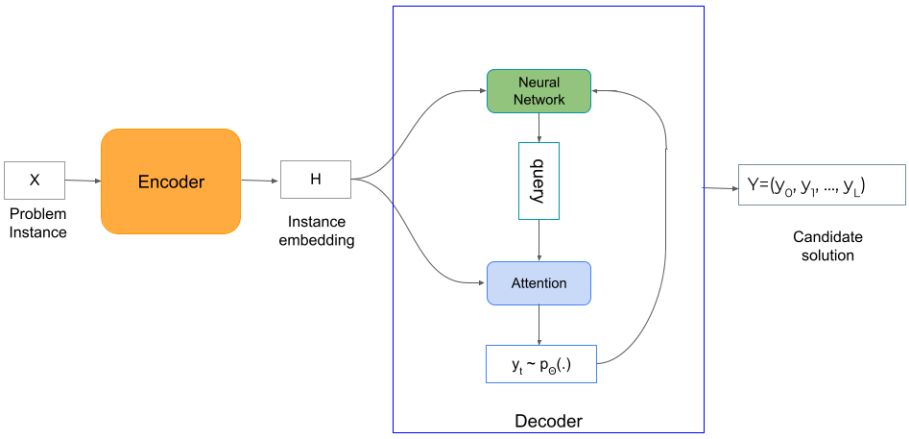

Fig. 1. The general encoder-decoder framework used to solve routing problems. The encoder takes as input a problem instance X and outputs an alternative representation H in an embedding space. The decoder iteratively constructs the candidate solution Y by adding a client or a depot y_t at each step t until all clients are visited.

3.2 Two-step algorithms for the vehicle routing problem

Classical two-step construction approaches for solving the CVRP involve (i) partitioning the clients into feasible clusters with regard to vehicle capacity and (ii) ordering them into routes of minimum length. Based on how the two operations are orchestrated, we can distinguish two types of two-step algorithms: Cluster-first Route-second and Order-first Split-Second.

In Cluster-first Route-second algorithms, the clients are first grouped together following the vehicle capacity constraint, then a traveling salesman problem is solved for each cluster using an

[3] $softmax(s_i) = \frac{exp(s_i)}{\sum_{j=1}^{K} exp(s_j)}$

exact solver or heuristics. The Sweep algorithm is the most common algorithm of this type [12]. Feasible clusters are constructed by considering the polar angle between the clients and the depot, then for each cluster a TSP is solved. An extension of this algorithm called the petal algorithm considers generating several routes and selects the final routes of the solution by solving a set partitioning problem [13]. Another work considers obtaining the clusters by solving a generalized assignment problem [14]. One major drawback of this approach is that it is not computationally efficient due to the clustering algorithms [15].

On the other hand, Order-first Split-second algorithms consider first ordering the customers into a sequence called a giant tour, to then, decompose it into a set of feasible tours considering the vehicle capacity. Traveling salesman problem heuristics are used to get giant tours, and the CVRP tours can be obtained optimally from the giant tours by solving a shortest path problem, as we will detail later. The first documented approach of this type generates the giant tour by random permutation of clients' visit order, followed by a 2-opt improvement, and then builds the routes using Floyd's algorithm [7]. Prins proposed the first genetic algorithm for the CVRP that relies on the Order-first Split-second approach, which was competitive with the best metaheuristic at that time (Tabu Search) [8]. In their approach, the authors proposed a representation of the chromosomes as giant tours and introduced the *Split* procedure based on an auxiliary acyclic graph generated on top of a giant tour. Bellman's algorithm is used in order to extract the feasible routes. HGS, today's state of the art metaheuristic for the CVRP, also uses a giant tour representation and the Split algorithm [16].

The Order-first Split-second approach is appealing. A recent review of this approach surveys more than 70 research papers that build heuristics and metaheuristics to successfully solve vehicle routing problems [17]. Computationally, it is less expensive to build a giant tour and then to split it than building clusters of clients. Also, the search space is reduced to the space of giant tours instead of the direct solution representations with depot placement. As highlighted in the survey, this search space reduction does not make the optimal solution unattainable, since there is an *optimal* giant tour which corresponds to the optimal solution. In addition, for a given giant tour, only its optimal split is retained. This ensures to prevent too many poor quality solutions from appearing often.

3.3 Graph Neural Networks

Since CVRP instances can be modelled as a graph, it is interesting to use neural networks that takes advantage of this structure. This makes Graph Neural Networks (GNNs) an ideal choice to compute a representation of an instance that captures useful information for the resolution process. We define a GNN by stacking K GNN blocks. Each block k relies on message passing in order to compute the node embeddings h_u^k, $\forall u \in V$. This mechanism can be viewed as a differentiable function that computes node embeddings as follows: $h_u^k = F(h_u^{k-1}, \{h_v^{k-1}\}_{v\in\mathcal{N}(u)}, \{e(u,v)\}_{v\in\mathcal{N}(u)})$, with $\mathcal{N}(u)$ being the set of the neighbor nodes of a node $u \in V$ and $\{e(u,v)\}_{v\in\mathcal{N}(u)}$ the set of edges that link the node u to its neighbors $v \in \mathcal{N}(u)$. We use the instance features as an initial input of the first GNN block. The function F itself relies on two mechanisms: neighborhood message aggregation and node embedding update, defined as:

$$m_u^{(k)} = \textsc{Aggregate}\big(\{h_v^{(k-1)}\}_{v\in\mathcal{N}(u)}, \{e(u,v)\}_{v\in\mathcal{N}(u)}\big)$$
$$h_u^{(k)} = \textsc{Update}(h_u^{(k-1)}, m_u^{(k)})$$

Aggregation can either be the mean, the maximum or the sum of neighbors' node embeddings. It can also be a weighted sum with weights computed using an attention mechanism [18]. It can take into consideration the edge weights of the neighboring nodes $e(u,v)$. The update function is a deep neural network that computes a new node embedding by using the message from the aggregation and the node embedding from the preceding block. Graph neural network models differ depending on the choice of the Aggregate and the Update functions.

We can distinguish two families of GNNs: spectral and spatial GNNs. Spectral GNNs rely on spectral graph representations based on graph signal processing theory, such as GCN [19]. Spatial GNNs, such as GAT [18], exploit the graph topology. Refer to Zhou et al. for a GNN review [20].

In the next section, we describe how we use the *Split* algorithm along with the NCO framework to train GNN models for solving the CVRP.

4 The Neural Order-first Split-second algorithm

As mentioned in the previous section, actual NCO construction-based policies for the CVRP produce a sequence by routing the clients and choosing when to return to the depot iteratively until all clients are served. These policies may lead to more returns to depot than necessary and produce poor quality solutions. For example, a policy can decide to refill in the depot after serving each client even if the vehicle capacity allows for serving more than one client at once. Learning from poor quality solutions can slow down and hamper the learning process and produce suboptimal policies. Instead of this, we propose to let the deep neural network build an indirect solution representation via the construction of the giant tour and to delay the routes construction to the Split algorithm. Thus, our neural network implicitly learns to solve vehicle routing problem instances by exploring the space of giant tours. Alternatively, we can view the neural network's output as a permutation of the clients' visit order, which is close to what is done in works for the TSP [3, 4]. This also simplifies the masking procedure used to avoid the appearance of a client twice in the solution. Another advantage of this approach is that the neural network can learn different policies depending on the variant of the vehicle routing problem (e.g. Capacitated VRP, VRP with Time Windows) without additional adaptation. The Split algorithm will handle the additional constraints, and the neural network learns the policy accordingly. Unlink other learning-based construction approaches that build a solution in a variable number of steps due to the return to the depot to refill, our neural network builds the giant tour in a fixed number of steps equal to the number of clients in the instance. Algorithm 1 presents the general approach that will be detailed afterwards.

For a given instance X of the CVRP, our neural network defines a stochastic policy that outputs the probability of generating a giant tour as a sequence Y. Using the probability chain rule, and with θ the parameters of the neural network, this policy is defined as follows:

$$P_\theta(Y|X) = \prod_{t=0}^{n-1} p_\theta(y_t|y_0, ..., y_{t-1}, X)$$

After sampling a sequence Y from P_θ, Y is then transformed into feasible routes using the Split algorithm with regard to the vehicle's capacity constraint. The Split algorithm can be viewed as an oracle that evaluates the goodness of a giant tour by returning the associated solution's total travelled distance. This evaluation makes it possible to train our deep neural network via reinforcement learning. We define the loss as the expected tour lengths of the Y sequences evaluated by the Split algorithm, i.e. $\mathcal{L}(\theta) = \mathbb{E}_{X\sim\mathcal{D}, Y\sim P_\theta(.|X)}\big[\text{Split}(Y, X)\big]$. The objective is to find the best parameters θ that will output good quality sequences Y that would result on short tour lengths. For this, we rely on AdamW as a gradient descent optimizer during training. In order to compute the gradient of the loss, we use REINFORCE with Rollout baseline [5]:

$$\nabla_\theta \mathcal{L}(\theta) = \mathbb{E}_{X\sim\mathcal{D}, Y\sim P_\theta(.|X)}\Big[\Big(\text{Split}(Y, X) - b(X)\Big)\nabla_\theta \log P_\theta(Y|X)\Big]$$

The gradient $\nabla_\theta \mathcal{L}(\theta)$ is approximated using Monte Carlo sampling over a batch of B i.i.d CVRP instances as follows:

$$\nabla_\theta \mathcal{L}(\theta) \approx \frac{1}{B}\sum_{i=1}^{B}\Big[\Big(\text{Split}(Y_i, X_i) - b(X_i)\Big)\nabla_\theta \log P_\theta(Y_i|X_i)\Big]$$

The baseline $b(X)$ is used to reduce the gradient variance, leading to an acceleration of the learning process. We use the greedy rollout baseline $b(X) = \text{Split}(Y^{BL}, X)$ which is an evaluation of the optimal Split of the giant tour Y^{BL} resulting from a copy of the learning neural network with parameters θ^{BL} that acts greedily, i.e. it chooses the next client with the highest probability of appearance at each time step. This baseline proved to be more efficient than actor-critic or REINFORCE with an exponential moving average baseline [5]. During validation, if the performance of θ is significantly better than that of θ^{BL} according to a t-test ($\alpha = 5\%$), the baseline is updated with the parameters of P_θ, i.e. θ^{BL} is set to θ.

4.1 Instance features

For each instance X, we define the nodes and edges features as follows:

Algorithm 1.1: NOFSS REINFORCE with Rollout Baseline

Inputs: θ, Number of epochs E, batch size B, number of instances K, number of clients n, vehicle capacity C, t-test threshold α
$T \leftarrow \frac{K}{B}$
$\theta^{BL} \leftarrow \theta$
for $e \leftarrow 1$ *to* E **do** `// train for` E `epochs`
  for $t \leftarrow 1$ *to* T **do** `// loop over the` T `instance batches`
    `// Get a batch of` B `CVRP instances with` n `clients`
    $X_i \leftarrow \text{getInstance}(n, C), \quad \forall i \in \{1, ..., B\}$
    `// Sample a giant tour according to the learning policy` P_θ
    $Y_i \leftarrow \text{SampleGiantTour}(X_i, P_\theta), \quad \forall i \in \{1, ..., B\}$
    `// Generate a giant tour greedily according to the policy` $P_{\theta^{BL}}$
    $Y_i^{BL} \leftarrow \text{GreedyGiantTour}(X_i, P_{\theta^{BL}}), \quad \forall i \in \{1, ..., B\}$
    `// Evaluate giant tours total travel cost`
    $L_i \leftarrow \text{Split}(X_i, Y_i, C) \quad \forall i \in \{1, ..., B\}$
    $L_i^{BL} \leftarrow \text{Split}(X_i, Y_i^{BL}, C) \quad \forall i \in \{1, ..., B\}$
    `// Compute the loss and update the neural network parameters`
    $\nabla_\theta \mathcal{L} \leftarrow \frac{1}{B} \sum_{i=1}^{B} (L_i - L_i^{BL}) \nabla_\theta \log P_\theta(Y_i|X_i)$
    $\theta \leftarrow AdamW(\theta, \nabla_\theta \mathcal{L})$
  end
  if *t-test*$(P_\theta, P_{\theta^{BL}}) < \alpha$ **then**
    $\theta^{BL} \leftarrow \theta$
  end
end

Node features. Each node $u \in V$ is represented as a quadruplet $(x_u, y_u, \hat{d}_u, a_u)$ where (x_u, y_u) are the node coordinates sampled from a uniform distribution $\mathcal{U}([0,1] \times [0,1])$, $\hat{d}_u = d_u/C \in [0,1]$ is the normalized demand and $a_u = atan((y_u - y_0)/(x_u - x_0)) \in]-\pi/2, \pi/2[$ is the polar angle between the node u and the depot node 0.

Edge features. For each edge $(u, v) \in E$, we define the edge features as the Euclidean distance between the nodes u and v (i.e. $d(u, v) \coloneqq \|u - v\|, \forall (u, v) \in E$). The distance between two nodes in the instance is an interesting feature in the case of vehicle routing problems, since it is information that characterizes the problem well, and it appears in the objective function.

4.2 NOFSS Encoding-Decoding architectures

The NOFSS approach is agnostic to the choice of the encoding and decoding model architectures. Thus, we propose to train various encoder-decoder models that rely on different graph neural networks (GNNs) and a GRU recurrent cell for decoding. The decoded sequence is passed to the Split algorithm in order to retrieve a candidate solution for the instance (Figure 2).

Encoding. We experiment three GNN Encoders for our approach: GCN (a spectral GNN), GAT (a spatial GNN) and TransformerConv (a spatial GNN) [21]. Each encoder have K similar blocks. The GNN outputs an embedding for each node (clients and depot) $h_u^{(K)} \in \mathbb{R}^d, \forall u \in V$ and a graph representation computed using an average pooling $\bar{h} = 1/|V| \sum_{u \in V} h_u^{(K)}$. Finally, to distinguish the clients embeddings from the depot embedding $h_0^{(K)}$, we pass them into a feedforward layer $h_u = W_c \cdot h_u^{(K)} + b_c, \forall u \in V - \{0\}$, with $W_c \in \mathbb{R}^{d \times d}, b_c \in \mathbb{R}^d$ being respectively the weights and the bias of the layer.

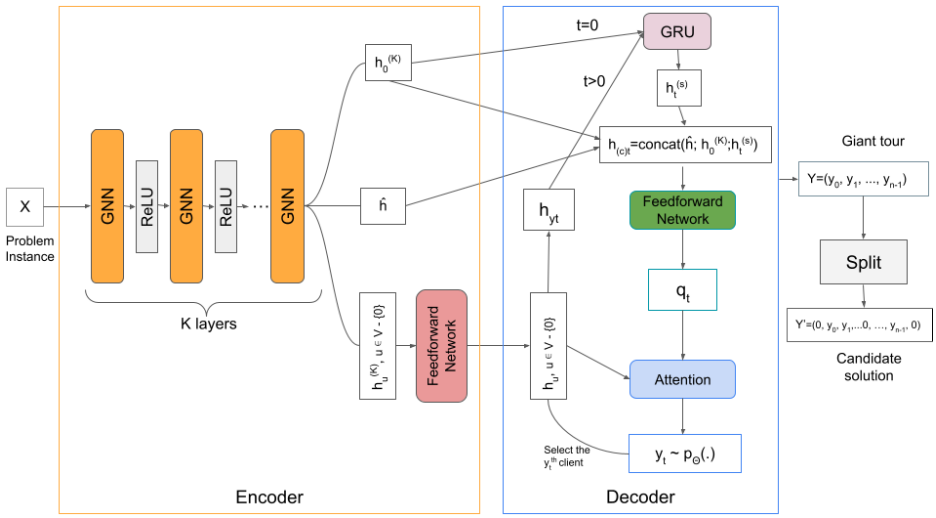

Fig. 2. Our proposed NOFSS model for solving CVRP instances.

Neighborhood definition. As highlighted in Section 2, we can define a CVRP instance as a complete graph. We define the neighborhood $\mathcal{N}(u)$ of a client node $u \in V - \{0\}$ as the κ nearest nodes in terms of Euclidean distance and the depot 0, since it is important for the client's representation to be aware of the depot's existence (i.e. $\mathcal{N}(u) = \{v_1, v_2, ..., v_\kappa \in V; \|v_1 - u\| \leq \|v_2 - u\| \leq ... \leq \|v_\kappa - u\|\} \cup \{0\}$). For the depot, we consider that it is connected to every client. An example of an instance neighborhood definition is depicted in Figure 3. The central node (red square) represents the depot, while the other nodes (blue circles) represent the clients. An edge exists between nodes u and v if $v \in \mathcal{N}(u)$. The number of nearest neighbors κ is determined per instance. We set it to be the average number of clients per route as if they were uniformly distributed on the routes, i.e. $\kappa = \frac{n}{m}$ with n being the number of clients and m being the lower bound of the number of routes. m is determined as the sum of all clients' demands divided by the vehicle's capacity rounded to the next integer ($m = \left\lceil \frac{\sum_{i=1}^{n} d_i}{C} \right\rceil$). The advantage of such a definition of κ is that it takes into account the characteristics of the instance in terms of the number of clients, their demands, and the capacity of the vehicles instead of selecting an arbitrary number of neighbors.

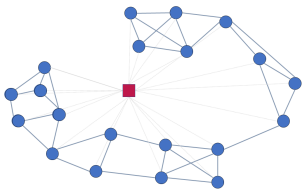

Fig. 3. CVRP instance with relationships between neighboring nodes (central square node is the depot).

Decoding. Since we are decoding a sequence of clients' order, we use a GRU recurrent cell [22]. GRU is relevant as it enables capturing the sequence representation while taking into account the order of its elements. It takes as input the previously selected client representation at step $t-1$ concatenated with the depot representation $h_0^{(K)}$ and incorporates it in the global representation

of the partial giant tour. At $t = 0$, we only use the depot representation $h_0^{(K)}$ as input to the GRU.

$$h_t^{(s)} = \begin{cases} \text{GRU}(h_0^{(K)}), & t = 0 \\ \text{GRU}([h_{y_{t-1}}; h_0^{(K)}]), & t > 0 \end{cases}$$

The graph embedding $\hat{h}$, the depot embedding $h_0^{(K)}$ and the sequence embedding $h_t^{(s)}$ are then concatenated together to form a context vector $h_c \in \mathbb{R}^{3d}$. The context vector is then passed to a feedforward layer made of two linear layers with $ReLU$ activation function in between to output a query vector $q_t \in \mathbb{R}^d$ i.e. $q = W_2 \cdot ReLU(W_1 \cdot h_c + b_1) + b_2$ with $W_2 \in \mathbb{R}^{d \times 3d}$, $W_2 \in \mathbb{R}^{d \times d}$, $b_1, b_2 \in \mathbb{R}^d$ being the parameters of the feedforward layer.

To compute the probability of selecting the next client $p_\theta(y_t | y_0, ..., y_{t-1}, X)$, we compute attention scores s_u ($\forall u \in V - \{0\}$) using a scaled dot-product with a masking mechanism in order to avoid selecting the same client twice. These scores are then clipped within $[-10, 10]$ using $tanh$ [5].

$$s_u = \begin{cases} c \cdot tanh\left(\dfrac{q_t h_u^\top}{\sqrt{d}}\right), & u \neq y_{t'} \quad t' < t, c = 10 \\ -\infty & \text{otherwise} \end{cases}$$

The attention scores are converted into a probability distribution using the softmax function $p_i = p_\theta(y_t = i | y_0, ..., y_{t-1}, X) = softmax(s_i)$ By setting the value of the attention score to $-\infty$, we can perform the masking of already visited clients. Thus, when passed to the softmax function, its associated probability will be 0.

The Split procedure. The algorithm works on the basis of the giant tour output by the neural network augmented with the depot, i.e $\mathcal{Y} = (y_0, y_1, ..., y_n)$ with $y_0 = 0$ being the depot. Using the giant tour, we define an auxiliary graph $H(V^H, E^H)$ with $|V^H| = n + 1$. The nodes in V^H indicate the depot (either for return or departure). The edge set indicates all possible sub-sequences that starts from y_i to y_j $(y_i, y_{i+1}, ..., y_j)$ that do not transgress the vehicle's capacity constraint. We formulate it as follows: $E^H = \{(i, j) \in V^H \times V^H; \quad i < j, \quad \sum_{k=i+1}^{j} d_{y_k} \leq C\}$. The edges are weighted as follows: for an edge $(i, j) \in E^H$ we associate the total travelled distance starting from the depot to the client y_{i+1}, visiting the tour $(y_{i+1}, ..., y_j)$ and going back to the depot from y_j:

$$D^H = \{d_{ij} = dist(0, y_{i+1}) + \sum_{\substack{k=i+1 \\ j-i>1}}^{j-1} dist(y_k, y_{k+1}) + dist(y_j, 0), \quad \forall (i, j) \in E^H\}$$

This gives us a direct acyclic graph where we solve a shortest path problem using Bellman's algorithm. The associated shortest path cost represents the best solution length (total travelled distance) for the CVRP instance with regard to the given giant tour.

5 Experiments

Data generation. We follow the data generation protocol of Nazari et al. [6] to consider 3 types of CVRP instances with number of clients $n = 20$, 50 and 100. For each problem size, we have generated $100k$ instances for training, and two sets of $10k$ instances for validation and test. Clients and depot locations are generated from a uniform distribution $\mathcal{U}(\{[0, 1] \times [0, 1]\})$. The clients' demands are also uniformly drawn from the interval $[1, 9]$. Vehicles' capacities are set to 30, 40 and 50 respectively for $n = 20$, 50, 100.

Hyperparameters. We use an embedding dimension $d = 128$ and a uniform parameter initialization for our deep neural networks $\mathcal{U}(-1/\sqrt{d}, 1/\sqrt{d})$ and set the learning rate to $\eta = 10^{-3}$. The models are trained with a time limit of 100 hours and batch size $B = 128$ on a single NVIDIA V100 GPU with 16 GB of VRAM. For each encoder type, we use $K = 3$ GNN blocks. Implementations use PyTorch and PyTorch Geometric for graph neural networks [23] (Python), while the Split algorithm is implemented in C.

Baselines. We use HGS[4] [16] as baseline as it is one of the state of the art metaheuristics for the CVRP. We also use classical CVRP heuristics[5]: (i) RFCS [7] as a two-step order-first split-second heuristic, (ii) Sweep [12] as a two-step cluster-first route-second approach, and (iii) Nearest Neighbor heuristic as a single-step construction approach [24]. We also trained the model with TransformerConv encoder in an end-to-end manner for depot and clients choice (Full-learning). We first note that NOFSS models are faster to train, completing $E = 1000$ of learning epochs in the 100 hours time budget, while the Full-learning models perform 1000, 500 and 200 training epochs for instance sizes of 20, 50 and 100 respectively. For the exploitation of the learned policies, we use a greedy decoding which considers the highest probability at each decoding step and a sampling strategy which samples 1280 candidate solutions for each test instance from the probability distributions given by the models. Table 1 reports the results of each approach on the test specifying: average solution lengths (obj.), the average gap (in percentage) to the best average solution lengths and the running time (in seconds) to output a candidate solution for a single instance.

Table 1. NOFSS vs. other algorithms. FL for Full-Learning; exploitation, greedy (G), sampling (S).

Method	$n = 20$			$n = 50$			$n = 100$		
	obj.	gap (%)	time (s)	obj.	gap (%)	time (s)	obj.	gap (%)	time (s)
HGS	6.13	0.00	0.003	10.34	0.00	0.09	15.57	0.00	0.69
RFCS	6.30	2.76	0.02	10.90	5.39	0.57	16.62	6.73	7.53
Sweep	7.55	23.16	0.01	15.60	50.93	0.06	28.56	83.37	0.23
Nearest neighbor	7.39	20.57	0.0004	12.63	22.19	0.001	18.95	21.68	0.01
NOFSS-GCN (G)	6.83	11.41	0.0008	12.31	19.05	0.003	19.41	24.66	0.007
NOFSS-GAT (G)	6.59	7.50	0.006	11.74	13.53	0.02	18.34	17.80	0.05
NOFSS-Transformer (G)	6.50	6.03	0.006	11.57	11.89	0.02	18.13	16.44	0.06
FL-Transformer (G)	6.49	5.87	0.006	11.34	9.67	0.02	17.69	13.61	0.06
NOFSS-Transformer (S)	6.24	1.79	1.37	11.03	6.67	1.56	17.45	12.07	2.43
FL-Transformer (S)	6.18	0.81	2.09	10.79	4.35	2.35	17.32	11.23	8.29

5.1 Comparison with a Full-learning setting

Figure 4 presents the evolution of the average solution length per epoch during training and validation on CVRP instances with 20 clients (left) and 50 clients (right). During training, candidate solutions are sampled from the model and their total lengths are averaged over the training set. Let us note that the models' parameters are updated each time a batch is processed via gradient descent, thus the performance of the models changes every batch during training, while validation is performed using the model resulting from the processing of the last batch in the training set, which is theoretically the best model achieved at the end of the epoch. Also, in validation, we use a greedy decoding instead of sampling. The evolution of the average solution lengths shows that the NOFSS model is able to learn an implicit policy for solving the CVRP by learning to output an indirect representation of the solution. On instances with 20 clients, we can observe that during training, the NOFSS model achieves better average solution lengths than the Full-learning model. On validation, we observe the same trend as in training, but starting from the 600^{th} epoch, the Full-learning model slightly outperforms the NOFSS model. The equivalent performance of the two models is confirmed on the test set with average solution lengths of 6.50 and 6.49 on greedy decoding for NOFSS and Full-learning respectively with similar execution times. On sampling decoding, similar performances are observed, with 0.9 % difference in performance between the two models, but with an advantage in execution time in favor of NOFSS. On CVRP with 50 clients, we observe that NOFSS has a better jump start performance on training and a better final performance for the Full-learning model. We observe 2 % difference in performance for greedy and sampling decoding on the test set. We also note similar sampling times for the two types of models in greedy decoding, while NOFSS being 52 %, 50% and 241% faster in sampling respectively for $n = 20$, 50 and 100.

[4] https://github.com/vidalt/HGS-CVRP
[5] https://github.com/yorak/VeRyPy

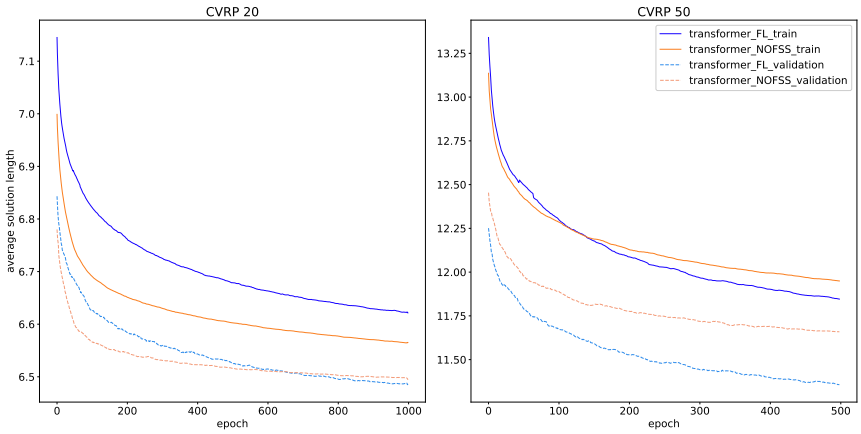

Fig. 4. Learning curves in training and validation for Full-learning (blue) and NOFSS models (orange) on CVRP instances with 20 (CVRP20) and 50 clients (CVRP50); lower is better.

5.2 Comparison to handcrafted heuristics

When compared to handcrafted heuristics, we can observe from Table 1 that either with greedy or sampling exploitation, NOFSS models outperform the Sweep and Nearest neighbor algorithms. NOFSS model seems to output better solution lengths, on average, than RFCS on CVRP with 20 clients when using the sampling strategy but seems to fail scaling to CVRP with 50 and 100 clients. Let us note that while RFCS and NOFSS belong to the same type of two-step strategy, there is a difference in the two approaches in that RFCS explicitly solves a Traveling Salesman Problem, while NOFSS directly evaluates the giant tour using the Split algorithm. The difference in average solution lengths may suggest that NOFSS learned policy is different from a policy that learns to solve a Traveling Salesman Problem.

5.3 Influence of the type of encoder

We investigate the influence of the choice of GNN encoder on models' performance. Figure 5 shows the evolution of the average solutions lengths per epoch in training and validation phases for the 3 types of GNN encoders: GCN, GAT and TransformerConv on CVRP with 20 and 50 clients. We observe the same trends for both training and validation phases, with TransformerConv having the best convergence, followed by GAT encoder and finally by GCN encoder. The instances' representation plays an important role in the resolution process, because a good representation leads to the exploitation of meaningful features and, thus, gives a better solution. The choice of the encoder seems to be a critical part of the model's architecture. It appears from these results that spatial GNNs better perform than spectral GNNs in our evaluation setting. Exploiting the graph topology in the spatial domain seems to benefit more in the context of vehicle routing problems than exploiting the graph structure in the spectral domain. While TransformerConv and GAT are both spatial GNNs, it seems that the way they exploit the node and edges information has an impact on the overall performance of the models.

5.4 On models generalization

We propose to study the generalization of the models trained on a set of instances with a specific size to instances of different size. For this, we evaluate the different test sets on instances of different sizes. For example, we evaluate the NOFSS Transformer model trained on CVRP with 20 clients instances (Transformer-20) on instances with 20, 50 and 100 clients. Table 2 sums up our results. We report the average solution lengths for both greedy and sampling exploitation strategies. For greedy decoding, we report the results for the models trained on the different instance sizes while for sampling, we focus on the model trained on instances sizes which seems more promising based on our findings on the greedy decoding. We observe that for the Transformer-20, the NOFSS model

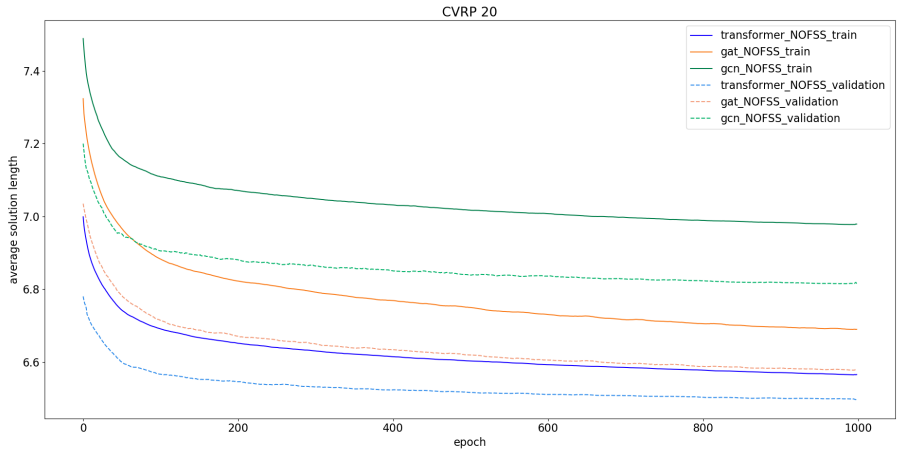

Fig. 5. Comparison of Graph Neural Network encoders on models' performance (training and validation).

has a better generalization property than the Full-learning model, with performance similar for $n = 20$ and $n = 100$ and better for $n = 50$. Since training models on instances with 20 clients is faster, it is relevant to identify that the NOFSS model is a better choice.

For Transformer-50 and Transformer-100, it appears that, for $n = 20$ NOFSS models have better performances than their Full-learning counterparts while staying competitive for $n = 50$ and $n = 100$. An interesting result observed on Transformer-50 is its good generalization to CVRP instances with 100 clients, as it appears that it achieves better performance than the models trained on instances with 100 clients. This may suggest that relevant invariants that are beyond the instance size are learned while training on instances with 50 clients. We push further our investigations on Transformer-50 by analyzing its performance with a sampling exploitation strategy. While for the instances with 20 clients, the models stay competitive with the ones trained on that size, they achieve the best performances on the sets with instances with 50 and 100 clients. Transformer-50 appears to be a good trade-off between learning speed (it is faster to train than Transformer-100) and performance.

Table 2. Comparison of average solution lengths achieved by the NOFSS and Full-learning models on different instance sizes of the test set.

Trained model	NOFSS (G)			Full-learning (G)		
	20	50	100	20	50	100
Transformer-20	6.50	11.62	18.34	6.49	12.01	18.33
Transformer-50	6.64	11.57	17.97	6.76	11.34	17.52
Transformer-100	6.94	11.79	18.13	6.98	11.65	17.69
	NOFSS (S)			Full-learning (S)		
	20	50	100	20	50	100
Transformer-50	6.31	11.03	17.40	6.25	10.79	17.22

6 Conclusion

In this work, we proposed NOFSS, a two-step algorithm hybridizing a deep neural network model and an exact tour splitting procedure for the Capacitated Vehicle Routing Problem. To the best of our knowledge, this is the first model that proposes a hybridization between a deep neural network and a dynamic programming algorithm to successfully learn an implicit policy based on giant tour generation to solve the CVRP. We conducted extensive experiments on the proposed models with various Graph Neural Network encoders and compared them against classic CVRP heuristics and an end-to-end Full-learning model. Our results show that NOFSS is very competitive, even if it

currently does not surpass end-to-end full-learning approaches. NOFSS is however faster than end-to-end approaches in both training and evaluation. It also shows good generalization properties when trained on instances with a specific size and applied to solve instances of different sizes. The NOFSS model is easier to implement than an end-to-end learning-based policy and does not rely on sophisticated handcrafted search strategies to find good quality solutions.

Future work should investigate more on the generalization of the method to instances of bigger sizes. Also, while we tested only greedy and sampling strategies for exploiting the trained models, other relevant strategies may be interesting such as beam search, or using bigger sample sizes than the one we used since NOFSS has a faster execution time. The solution given by NOFSS can also be a good warm start for further improvement by local search algorithms. Finally, since our approach is generic, it would be interesting to evaluate it on other problems, such as the Vehicle Routing Problem with Time Windows.

References

1. Bengio, Y., Lodi, A., Prouvost, A.: Machine learning for combinatorial optimization: a methodological tour d'horizon. European Journal of Operational Research **290** (2021) 405–421
2. Vinyals, O., Fortunato, M., Jaitly, N.: Pointer networks. arXiv:1506.03134 (2015)
3. Bello, I., Pham, H., Le, Q.V., Norouzi, M., Bengio, S.: Neural combinatorial optimization with reinforcement learning. arXiv:1611.09940 (2016)
4. Deudon, M., Cournut, P., Lacoste, A., Adulyasak, Y., Rousseau, L.M.: Learning heuristics for the tsp by policy gradient. In: International conference on the integration of constraint programming, artificial intelligence, and operations research, Springer (2018) 170–181
5. Kool, W., Van Hoof, H., Welling, M.: Attention, learn to solve routing problems! arXiv:1803.08475 (2018)
6. Nazari, M., Oroojlooy, A., Snyder, L.V., Takáč, M.: Reinforcement learning for solving the vehicle routing problem. arXiv:1802.04240 (2018)
7. Beasley, J.E.: Route first—cluster second methods for vehicle routing. Omega **11** (1983) 403–408
8. Prins, C.: A simple and effective evolutionary algorithm for the vehicle routing problem. Computers & operations research **31** (2004) 1985–2002
9. Toth, P., Vigo, D.: The vehicle routing problem. SIAM (2002)
10. Smith, K.A.: Neural networks for combinatorial optimization: a review of more than a decade of research. INFORMS Journal on Computing **11** (1999) 15–34
11. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in neural information processing systems. (2017) 5998–6008
12. Gillett, B.E., Miller, L.R.: A heuristic algorithm for the vehicle-dispatch problem. Operations research **22** (1974) 340–349
13. Ryan, D.M., Hjorring, C., Glover, F.: Extensions of the petal method for vehicle routeing. Journal of the Operational Research Society **44** (1993) 289–296
14. Fisher, M.L., Jaikumar, R.: A generalized assignment heuristic for vehicle routing. Networks **11** (1981) 109–124
15. Hiquebran, D., Alfa, A., Shapiro, J., Gittoes, D.: A revised simulated annealing and cluster-first route-second algorithm applied to the vehicle routing problem. Engineering Optimization **22** (1993) 77–107
16. Vidal, T.: Hybrid genetic search for the cvrp: Open-source implementation and swap* neighborhood. Computers & Operations Research **140** (2022) 105643
17. Prins, C., Lacomme, P., Prodhon, C.: Order-first split-second methods for vehicle routing problems: A review. Transportation Research Part C: Emerging Technologies **40** (2014) 179–200
18. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y.: Graph attention networks. arXiv:1710.10903 (2017)
19. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv:1609.02907 (2016)
20. Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., Sun, M.: Graph neural networks: A review of methods and applications. AI Open **1** (2020) 57–81
21. Shi, Y., Huang, Z., Feng, S., Zhong, H., Wang, W., Sun, Y.: Masked label prediction: Unified message passing model for semi-supervised classification. arXiv:2009.03509 (2020)
22. Cho, K., Van Merriënboer, B., Bahdanau, D., Bengio, Y.: On the properties of neural machine translation: Encoder-decoder approaches. preprint arXiv:1409.1259 (2014)
23. Fey, M., Lenssen, J.E.: Fast graph representation learning with PyTorch Geometric. In: ICLR Workshop on Representation Learning on Graphs and Manifolds. (2019)
24. Rasku, J., Kärkkäinen, T., Musliu, N.: Meta-survey and implementations of classical capacitated vehicle routing heuristics with reproduced results. Toward Automatic Customization of Vehicle Routing Systems (2019)