9-2024

Efficient neural collaborative search for pickup and delivery problems

Detian KONG

Yining MA

Zhiguang CAO
Singapore Management University, zgcao@smu.edu.sg

Tianshu YU

Jianhua XIAO

Efficient Neural Collaborative Search for Pickup and Delivery Problems

Detian Kong*, Yining Ma*, Zhiguang Cao, Tianshu Yu, and Jianhua Xiao†.

Abstract—In this paper, we introduce Neural Collaborative Search (NCS), a novel learning-based framework for efficiently solving pickup and delivery problems (PDPs). NCS pioneers the collaboration between the latest prevalent neural construction and neural improvement models, establishing a collaborative framework where an improvement model iteratively refines solutions initiated by a construction model. Our NCS collaboratively trains the two models via reinforcement learning with an effective shared-critic mechanism. In addition, the construction model enhances the improvement model with high-quality initial solutions via curriculum learning, while the improvement model accelerates the convergence of the construction model through imitation learning. Besides the new framework design, we also propose the efficient Neural Neighborhood Search (N2S), an efficient improvement model employed within the NCS framework. N2S exploits a tailored Markov decision process formulation and two customized decoders for removing and then reinserting a pair of pickup-delivery nodes, thereby learning a ruin-repair search process for addressing the precedence constraints in PDPs efficiently. To balance the computation cost between encoders and decoders, N2S streamlines the existing encoder design through a light Synthesis Attention mechanism that allows the vanilla self-attention to synthesize various features regarding a route solution. Moreover, a diversity enhancement scheme is further leveraged to ameliorate the performance during the inference of N2S. Our NCS and N2S are both generic, and extensive experiments on two canonical PDP variants show that they can produce state-of-the-art results among existing neural methods. Remarkably, our NCS and N2S could surpass the well-known LKH3 solver especially on the more constrained PDP variant. Our code is available at: https://github.com/dtkon/PDP-NCS.

Index Terms—Learning to optimize, deep reinforcement learning, attention mechanism, pickup and delivery, neighborhood search

1 Introduction

The pickup and delivery problem (PDP) involves finding the shortest route to fulfil a set of pickup and delivery orders, and is relevant to applications such as logistics, robotics, and food delivery [1]. Unlike other routing problems, such as the travelling salesman problem (TSP) or the capacitated vehicle routing problem (CVRP), PDPs are featured by the precedence constraints that require each pickup to undergo before its ego delivery, which introduces distinct challenges and complexities to the optimization. For decades, numerous neighborhood search heuristics, which consider iteratively transforming a solution into its neighboring solution, have been proposed for PDPs [2], [3]. Typically, their efficiency hinges on the design of the neighborhood and search rules. However, they are often manually engineered and problem-specific, which need to be redesigned when changes occur in constraints or objectives. For instance, introducing a Last-In-First-Out (LIFO) constraint on loading/unloading may render basic PDP heuristic solvers inefficient, necessitating new neighborhoods or rules particularly tailored for the LIFO variant [4]. This limitation hinders their applications to the rapidly evolving industry.

Recently, significant progress has been made in leveraging deep reinforcement learning (DRL) for solving vehicle routing problems (VRP) (e.g., [5], [6]). These solvers are usually faster and could automate the design of heuristics for various VRP variants by recognizing useful patterns from a distribution of VRP instances [6]. Their adaptable nature facilitates the development of solvers for different variants with minimal human intervention, i.e., we can leverage a single learning framework to train respective models for different VRP variants [7]. In general, DRL-based solvers are classified into *construction* and *improvement* ones. However, in our view, each of the two lines of methods exhibits advantages while still suffering drawbacks. Construction methods (e.g., [6], [7]) are faster in terms of solution construction but may require longer training time and face challenges in escaping local optima. Improvement methods (e.g., [5], [8]) are designed to learn the search process, but their efficiency could be hindered by exploring the entire large search space. Meanwhile, existing neural methods mainly focus on TSP or CVRP only, where efficient neural solvers for PDPs are not extensively studied. Despite the early attempt in [9], which learns a *construction* model to build a PDP solution (route) in seconds, it leaves a considerable gap to traditional heuristics in solution quality.

To address the above limitations and reduce the gap, we propose **Neural Collaborative Search (NCS)**, a hybrid framework that pioneers the collaboration between neural *construction* and neural *improvement* models for efficiently solving PDPs. Essentially, NCS operates as a collaborative system, where an improvement model iteratively refines solutions generated by a construction model. Consequently, NCS seeks to harness the rapid solution generation capability of construction methods to reduce

- † *Corresponding Author*
- * *Detian Kong and Yining Ma are equally contributed*
- *Detian Kong and Jianhua Xiao are with The Research Center of Logistics, Nankai University, China; Jianhua Xiao is also with the Laboratory for Economic Behaviors and Policy Simulation, Nankai University, China. (E-mails: kdt@mail.nankai.edu.cn, jhxiao@nankai.edu.cn)*
- *Yining Ma is with the Department of Industrial Systems Engineering and Management, National University of Singapore, Singapore. (E-mail: yiningma@u.nus.edu)*
- *Zhiguang Cao is with the School of Computing and Information Systems, Singapore Management University, Singapore. (E-mail: zhiguangcao@outlook.com)*
- *Tianshu Yu is with the School of Data Science, The Chinese University of Hong Kong, Shenzhen. (E-mail: yutianshu@cuhk.edu.cn)*

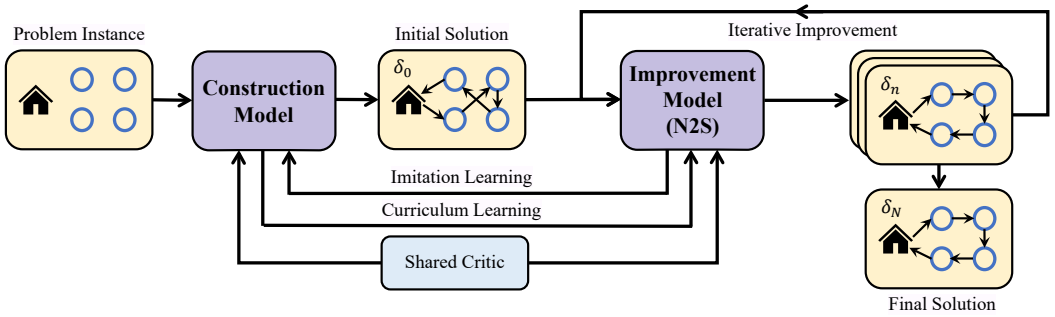


Fig. 1: The overall framework of NCS. Particularly, the imitation learning, curriculum learning and shared critic are elements of the training algorithm. Meanwhile, δ_0 denotes the initial solution constructed by the construction model, δ_n denotes the set of solutions that are being improved iteratively by the improvement model, and δ_N denotes the final best solution.

search space and exploit the search capabilities of improvement methods to efficiently escape local optima. As shown in Fig. 1, the two models are trained simultaneously and collaboratively through the actor-critic-based reinforcement learning [5] via a shared-critic mechanism, where the critic network of improvement model is shared to derive the critic of the construction model, saving much computation overhead. In addition, the construction model could enhance the improvement model through a curriculum learning (CL) strategy that yields suitable initial solutions as the starting point of the improvement model, thus facilitating search efficiency and better search scope towards more optimal regions. On the other hand, the improvement model enhances the construction model through an imitation learning (IL) loss to accelerate its training convergence. Our NCS demonstrates that the collaboration of two models could yield superior performance compared to the independent training or inference of individual models.

Besides the NCS framework, we also present **Neural Neighborhood Search (N2S)**[1], an efficient improvement model employed within the NCS framework. Our preliminary analysis finds that current transformer-styled improvement models [5], [8] face challenges in direct application to PDPs (details discussed in Appendix A). Their action space designs, such as *2-opt*, *insert*, or *swap* often fail to properly tackle the precedence constraints in PDPs. This leads to inefficient searches for solving large-scale or highly-constrained PDPs and often complicates DRL training by making a significant portion of the action space invalid. Conversely, our N2S tackles the precedence constraint efficiently by allowing a pair of pickup-delivery nodes to be simultaneously operated in the neighborhood search through two customized *removal* and *reinsertion* decoders as shown in Fig. 3. Moreover, as decoders become more sophisticated, achieving a computational balance between encoders and decoders is crucial. Meanwhile, it was revealed in [5] that the vanilla Transformer encoder [11] failed to correctly encode route solutions since the embeddings of node features (i.e., coordinates) and node positional features (i.e., node positions in a solution sequence) involve two different *aspects* of a route solution which are not directly compatible during encoding. They thus proposed the dual-aspect collaborative attention (DAC-Att) to learn dual representations for each feature *aspect*. In our N2S, we propose a simple yet powerful *Synthesis Attention (Synth-Att)* where the attention scores from various types of node feature embeddings can be synthesized to attain a comprehensive and informative representation. It not only has the potential to encode more *aspects* than DAC-Att, but also reduces the computation costs while reserving competitive performance.

Additionally, we design a diversity enhancement scheme to further ameliorate the performance. We evaluate the performance of our proposed NCS and N2S on two canonical problems in the PDP family, i.e., the pickup and delivery travelling salesman problem (PDTSP) and its variant with the LIFO constraint (PDTSP-LIFO). Experimental results show that our N2S, when used alone, could already outperform the state-of-the-art learning-based baselines, and become the first neural method to surpass the well-known LKH3 solver [12] on PDPs. Leveraging our proposed NCS collaborative framework further strengthens this superiority. More importantly, both N2S and NCS can use the same framework to tackle both PSTDP and PDTSP-LIFO without any human intervention, which is a significant improvement over existing hand-crafted heuristics requiring manual redesign.

The key contributions of this paper are threefold:

1) We introduce the NCS framework, pioneering the collaboration between neural construction and improvement models. Through the shared-critic mechanism, curriculum learning, and imitation learning designs, the collaboratively trained model achieves better results than when trained separately. NCS also holds potential applicability to other VRP variants.
2) We introduce the improvement model, N2S, marking the first instance of a neural ruin-repair search method tailored for PDPs. We also propose Synth-Att in N2S, allowing vanilla self-attention to synthesize attention relationships from various feature embeddings efficiently with superior expressiveness compared to the DAC-Attention.
3) We propose a diversity enhancement scheme, resulting in our NCS and N2S models achieving state-of-the-art performance. Notably, they are the first neural approaches with almost no domain knowledge to surpass the LKH3 solver on synthesized PDP instances, especially for the PDTSP-LIFO variant.

The reminders of this paper are as follows. Section 2 reviews various types of neural methods for VRPs including PDPs. Section 3 introduces the preliminaries. Section 4 details the architectures of the improvement model (N2S) and the construction model within NCS, alongside the methodologies for collaboratively training them and the inference algorithm. The experimental results and analysis are given in Section 5. Finally, Section 6 summarizes our work and points out the future work direction.

1. This article expands upon our previously published work presented at the 31st international joint conference on artificial intelligence (IJCAI), 2022 [10].

TABLE 1: Main features of the baseline neural methods.

Method	Type & Problem	Network Design	Algorithm Design
Heter-AM	Construction PDTSP	Heterogeneous Attention	REINFORCE with rollout baseline
POMO	Construction TSP & CVRP	Attention Model (AM)	REINFORCE with diverse rollouts
DACT	Improvement (2-opt) TSP & CVRP	Dual-Aspect Collaborative Transformer	n-step PPO actor-critic with CL
N2S	Improvement (**remove & reinsert**) PDTSP, PDTSP-LIFO	**Two decoders + Synth-Att**	n-step PPO actor-critic with CL
NCS	**Construction + Improvement** PDTSP, PDTSP-LIFO	Modified AM + **N2S**	n-step PPO actor-critic with **shared-critic** and **CL** and **IL**

2 Related Work

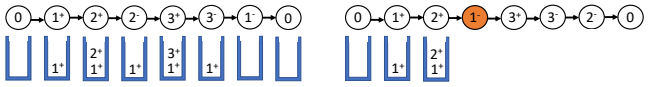
2.1 Neural Methods for VRPs

We classify recent *neural* methods into *construction* and *improvement* ones. The *construction* methods, e.g., [13], [14] and Attention Model (AM) [7], learn a distribution of selecting nodes to autoregressively build solutions from scratch. Despite being fast, they lack abilities to search (near-)optimal solutions, even if armed with sampling (e.g., [7]), local search (e.g., [15]), or Monte-Carlo tree search (e.g., [16]). Among them, POMO [6] which explored diverse rollouts and data augments is recognized as the best construction method. In addition to classical VRPs, the study on different variants and scales of VRPs has also aroused the interest of researchers. NHDE [17] constructs Pareto solutions for multi-objective VRPs with diversity enhancement. TAM [18] features a two-stage divide method to generate sub-route sequence to solve large-scale VRPs in real-time. Differently, *improvement* methods often hinge on a neighborhood search procedure such as *node swap* in [19], *ruin-and-repair* in [20], and *2-opt* in [8], and often rely on longer run time than construction methods. The work in [5] extended the Transformer styled model of [8] to Dual-Aspect Collaborative Transformer (DACT), and achieved state-of-the-art performance, which was also competitive to the *hybrid* neural methods, e.g., the ones combined with differential evolution [21] and dynamic programming [22]. NeuOpt [23] circumvents the pure feasibility masking scheme and learns to perform flexible k-opt exchanges, becomes the first work to enable the autonomous exploration of both feasible and infeasible regions. Different from construction and improvement methods, Difusco [24] incorporates a diffusion model that generates heatmaps for edge selection, yielding significant outcomes on the TSP.

Despite the success of the above methods for CVRP or TSP, they are not verified on the precedence-constrained PDPs. Though the first attempt was made in Heter-AM [9] to learn a construction solver for PDPs by introducing the heterogeneous attention to AM [7], the solution qualities are still far from the optimality. In this work, we take the representative construction methods POMO and Heter-AM, and improvement model DACT as our baselines. Table 1 lists the main differences between our methods and them.

2.2 Collaborative Neural Methods for VRPs

Recent studies highlight the potential of learning collaborative methods for VRPs, which integrate multiple independent methods to form a unified approach. The work in [25] proposed to combine the neural construction method with a traditional neighborhood search, which takes the output of the construction model as the initial solution for local search to yield the final VRP solution. The learning collaborative policies (LCP) in [15] considered dividing the search process into seed stage and revise stage for VRPs. Two different construction networks were employed to perform the respective tasks, with the initial solution constructed in the seed phase and (re)-constructed in the revised phase. Nevertheless, this method is still much less efficient than POMO aforementioned. In [26], a neural constructor and a neural perturbator were independently designed, which were directly stacked together during the inference. However, the above works failed to train the respective methods in an integrated and collaborative way, thus preventing effective information sharing and holding back the eventual performance. Furthermore, none of them is designed to tackle the pickup and delivery problems as studied in this paper.



(a) δ_1 (feasible to both PDPs) (b) δ_2 (only feasible to PDTSP)

Fig. 2: Two PDP solutions. (a) all goods are on top of the stack at the delivery node; (b) goods from 1^+ is blocked by 2^+ at 1^-.

2.3 Neighbourhood Search for PDPs

Various heuristics based on neighborhood search have been proposed for PDPs. For PDTSP, the *k-interchange* neighborhood was studied in [27]. A *ruin-and-repair* neighborhood was later proposed in [28], and further extended in [29] with multiple perturbation methods. Aside from PDTSP, other PDP variants were also investigated, normally solved by designing new problem-specific neighborhoods [1], e.g., five neighborhoods were proposed in [30] to tackle the PDTSP with handling costs. Among them, the PDTSP-LIFO attracts much attention due to its largely constrained search space. To solve it, additional neighborhoods such as *double-bridge* and *shake* were introduced in [31]. Neighborhoods with tree structures were further proposed in [4]. Different from the above PDP solvers, the well-known LKH3 solver [12] combines neighborhood restriction strategies to achieve a more effective local search, which could solve various VRPs with superior performance. Recently, LKH3 was extended to tackle several PDP variants including PDTSP and PDTSP-LIFO, and delivered comparable performance to [29] and [4] on PDTSP and PDTSP-LIFO, respectively. Also given the open-sourced nature[2], we use LKH3 as the benchmark heuristic.

3 Preliminary

Existing neural solvers mainly focus on TSP and CVRP. TSP involves finding the shortest possible route that visits a set of given nodes exactly once and returns to the starting node. CVRP is an extension of the TSP where a (fleet of) vehicle(s) departs from a central depot and must serve a set of customers while respecting vehicle capacity constraints and minimizing total travel distance (or cost). In contrast, despite sharing some characteristics with the TSP and CVRP, the PDP is significantly different from

2. http://webhotel4.ruc.dk/∼keld/research/LKH-3/

them due to the presence of pickup and delivery nodes. As we have verified in Appendix A, such precedence constraints would introduce significant challenges for existing neural solvers. We define the studied PDPs over a graph $G = (V, E)$, where nodes in $V = P \cup D \cup 0$ represent locations and edges in E represent connections between locations. Each node possesses a coordinate attribute and can be represented using the two-dimensional Euclidean coordinates. With n one-to-one pickup-delivery requests, an PDP instance contains $|V| = 2n+1$ different locations, where node 0 is depot, node set $P = \{1^+, 2^+, ..., n^+\}$ is referred to as pickup nodes, and node set $D = \{1^-, 2^-, ..., n^-\}$ is referred to as delivery nodes[3]. Each pickup node i^+ has a number of goods to be transported to its pairing delivery node i^-. The objective is to find the shortest Hamiltonian cycle to fulfil all requests. In this paper, we consider two representative PDP variants, i.e., PDTSP and PDTSP-LIFO. The solution δ is defined as a cyclic sequence $(x_0, ..., x_{2n+1})$, where x_0 and x_{2n+1} are the depot, and the rest is a permutation of nodes in $P \cup D$. The objective value of a solution is the total Euclidean length of the cyclical sequence of nodes. For PDTSP, such permutation is under the *precedence* constraint that requires each pickup i^+ to be visited before its pairing delivery i^-. For PDTSP-LIFO, the *last-in-first-out* constraint is further imposed which requires loading and unloading to be executed in the corresponding order. This implies that unloading at a delivery node is allowed if and only if the goods is at the top of the stack. In Fig. 2, we present two example solutions with $n = 3$ and $|V| = 7$. The two solutions are both feasible to PDTSP, however, solution δ_2 in Fig. 2(b) is infeasible to PDTSP-LIFO as the goods from 1^+ is *NOT* at the top of the stack when it needs to be delivered at 1^-. The mathematical formulations of PDTSP and PDTSP-LIFO are presented in Appendix B.

4 METHODOLOGY

Our Neural Collaborative Search (NCS) approach considers the collaboration of the two policy network, i.e., the improvement model and the construction model, respectively. The improvement model, referred to as the Neural Neighborhood Search (N2S) [10], and the construction model, a modified version of the Attention Model [7], named m-AM. These two models work collaboratively to solve the PDPs, with the construction model yielding the initial solution that the improvement model iteratively refines.

To achieve such collaboration, an intuitive way is to directly connect the two models that are trained independently. However, it would be far from optimal. On the one hand, the improvement model, typically initiating its search from a random solution, may perform less efficiently than starting from a solution generated by the construction model. On the other hand, the high-quality solutions produced by the improvement model, which could be beneficial for training the construction model, are often overlooked. In our NCS, we build a bridge between the construction model and improvement model to achieve collaborative training through a shared-critic mechanism, where they could naturally further promote each other with a curriculum learning strategy and an imitation learning loss function, respectively.

To better present our method, we first introduce the Markov Decision Process (MDP) formulation of N2S and m-AM (section 4.1). Following this, we detail the design of the policy network of N2S (section 4.2) and m-AM (section 4.3). Finally, we elaborate on how to train the construction model and the improvement model collaboratively in NCS (section 4.4). To simplify notation, some symbols used for m-AM may be identical to those used for N2S. For example, s denotes the state in the N2S model and is also used to represent the state in the m-AM model.

3. We also refer to x_0 as the depot, $\{x_1, ..., x_n\}$ as the pickup nodes, and $\{x_{n+1}, ..., x_{2n}\}$ as the delivery nodes from here on.

4.1 MDP Formulation

4.1.1 MDP Formulation of N2S

The input of N2S policy network is a problem instance and a solution to be improved, and the output is an action to improve the solution. Given the input and output of the policy network, we define the process of solving PDPs by our N2S as a Markov Decision Process $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$ as follows.

State $\mathcal{S}$. At time step t, the state is defined to include, 1) features of the current solution δ_t, 2) action history, and 3) objective value of the best incumbent solution, i.e.,

$$s_t = \{\{l(x)\}_{x \in V}, \{p_t(x)\}_{x \in V}, \mathcal{H}(t, K), f(\delta_t^*)\}, \quad (1)$$

where δ_t is described from two *aspects* following [5]: $l(x)$ contains 2-dim coordinates of node x (i.e., *node features*) and $p_t(x)$ indicates the index position of x in δ_t (i.e., *node positional feature*); $\mathcal{H}(t, K)$ stores the most recent K actions at time step t if any; and $f(\cdot)$ denotes the objective function and $\delta_t^* = \arg\min_{\delta_\tau \in \{\delta_0, ..., \delta_t\}} f(\delta_\tau)$.

Action $\mathcal{A}$. With action $a_t = \{(i^+, i^-), (j, k)\}$ where $j, k \in V \backslash \{i^+, i^-\}$, the agent removes node pair (i^+, i^-), and then reinserts node i^+ and i^- after node j and k, respectively.

State Transition $\mathcal{T}$. Performing action a_t on a solution will occur a deterministic transition and form a new solution. The action history and $f(\delta_t^*)$ will also be updated deterministically.

Reward $\mathcal{R}$. The reward function is defined as $r_t = f(\delta_t^*) - min\left[f(\delta_{t+1}), f(\delta_t^*)\right]$ which is the immediate reduced cost w.r.t. $f(\delta_t^*)$. The N2S agent aims to maximize the expected total reduced cost w.r.t. δ_0 with a discount factor $\gamma < 1$.

4.1.2 MDP Formulation of m-AM

The input of m-AM policy network is a problem instance and a partially constructed solution, and the output is an action to further construct the partial solution. Based on this, the MDP formulation of m-AM is defined as follows.

State $\mathcal{S}$. The state s_t represents the instance information and the partial solution constructed up to time step t. The former includes the coordinates of the depot and all customer nodes, denoted as $l(x)$ with $x \in V$, and the latter includes the set of nodes visited till time step t-1 and the one visited exactly at time step t-1. At time step 0, there is no last visited node, so $s_0 = \{l(x)\}_{x \in V}$.

Action $\mathcal{A}$. The action a_t of the construction model represents the next node to visit where the infeasible nodes will be filtered out.

State Transition $\mathcal{T}$. Based on a_t, the partial solution will append the newly visited node and lead to the next state s_{t+1}. The transition will be terminated when all nodes have been visited once, and the vehicle will return to the depot.

Reward $\mathcal{R}$. The goal is to construct a route with minimal travel length. Therefore, the reward of the whole action trajectory is set to the negative of the route length after the route is completed.

Policy $\mathcal{P}$. The stochastic policy $\pi_{\theta'}$ selects one node to visit (i.e., a_t) at each time step t. This process will be repeated until all customer nodes are served before returning to the depot. The final solution $\delta = (a_0, a_1, \ldots, a_{|V|-1}, a_0)$ can be described as a permutation of actions. Note that for PDPs, $a_0 = x_0$ is always

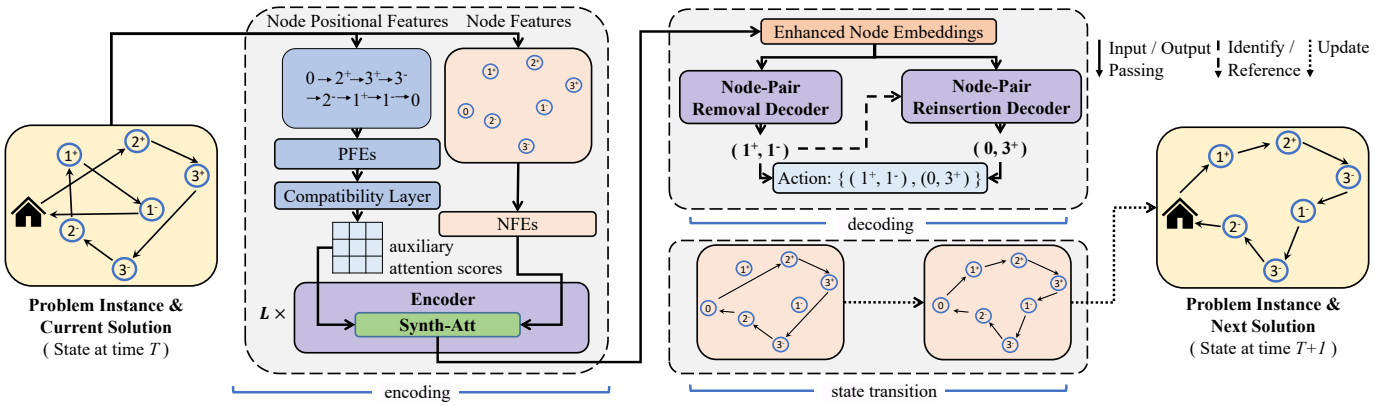


Fig. 3: N2S policy network (from the left to the right: encoding, decoding and state transition process)

the depot. The stochastic policy $\pi_{\theta'}$ for constructing a solution δ given a problem instance $\{l(x)\}_{x\in V}$ can be factorized and parameterized by θ' as

$$\pi_{\theta'}(\delta|\{l(x)\}_{x\in V}) = \pi_{\theta'}(\delta|s_0) = \prod_{\tau=0}^{|V|-1} \pi_{\theta'}(a_\tau|s_\tau). \quad (2)$$

4.2 Policy Network of N2S

N2S network is designed based on the encoder-decoder architecture, in which the encoder elegantly embeds the problem instance and current solution in sequence, and then the decoder adopts the compatibility computation to produce probability matrices of selecting node pairs to remove and reinsert. An example of a PDTSP-7 instance is shown in Fig. 3 to illustrate our N2S network. The encoder processes raw features of the current solution to produce node embeddings, which are then fed into the two decoders to sample an action. In the state transition, the node pair $(1^+, 1^-)$ is removed and then reinserted after depot 0 and node 3^+, respectively.

4.2.1 Encoder and Synth-Att

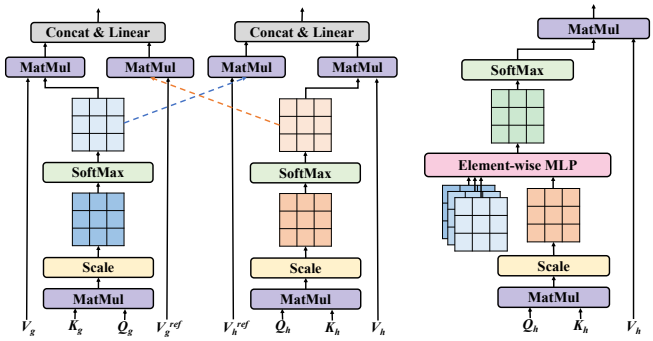
Given state $s = \{\{l(x)\}_{x\in V}, \{p(x)\}_{x\in V}, \mathcal{H}(t,K), f(\delta^*)\}$, the N2S encoder takes $\{l(x)\}_{x\in V}$ and $\{p(x)\}_{x\in V}$ as inputs[4] to learn embeddings for representing the current solution. Following DACT, we first project these raw features into two sets of embeddings, i.e., node feature embeddings (NFEs) $\{h_i\}_{i=0}^{|V|}$ and positional feature embeddings (PFEs) $\{g_i\}_{i=0}^{|V|}$. Different from [5], we treat NFEs as the primary set of embeddings whereas PFEs as auxiliary ones.

NFEs. We define h_i as the linear projection of its node features $l(x_i)$ for any $x_i \in V$ with output dimension $d_h = 128$.

PFEs. By extending the absolute positional encoding in the vanilla Transformer [11], the cyclic positional encoding (CPE) was proposed in [5], which enables Transformer to encode cyclic sequences (as our PDP solutions) more accurately. The PFE g_i with output dimension $d_g = 128$ is initialized by CPE as follows,

$$g_i{}^{(d)} = \begin{cases} sin(\omega_d \cdot \left|(z(i) \bmod \frac{4\pi}{\omega_d}) - \frac{2\pi}{\omega_d}\right|), & \text{if } d \text{ is even} \\ cos(\omega_d \cdot \left|(z(i) \bmod \frac{4\pi}{\omega_d}) - \frac{2\pi}{\omega_d}\right|), & \text{if } d \text{ is odd} \end{cases} \quad (3)$$

4. $\mathcal{H}(t,K)$ is the input to decoder and $f(\delta^*)$ is the input to critic network (introduced later). Here we omit time step t for better readability.



(a) The **DAC-Att** [5] (b) Our **Synth-Att**

Fig. 4: Comparison of attention modules for VRPs. The blue and orange squares are used to represent self-attention score matrices.

where superscript d of $g_i{}^{(d)}$ refers to the d-th dimension of g_i, and the scalar $z(i)$ as well as the angular frequency $\omega_d = \frac{2\pi}{T_d}$ are defined accorting to Eq. (4) and Eq. (5), respectively.

$$z(i) = \frac{i}{|V|}\frac{2\pi}{\omega_d}\left\lceil \frac{|V|}{2\pi/\omega_d} \right\rceil, \quad (4)$$

$$T_d = \begin{cases} \frac{3\lfloor d/3 \rfloor + 1}{d_g}(|V| - |V|^{\frac{1}{\lceil d_g/2 \rceil}}) + |V|^{\frac{1}{\lceil d_g/2 \rceil}}, & \text{if } d < \lfloor \frac{d_g}{2} \rfloor \\ |V|. & \text{otherwise} \end{cases} \quad (5)$$

According to [5], directly fusing the two sets of embeddings (i.e., $h_i + g_i$) may cause undesired noises to the vanilla self-attention. As shown in Fig. 4(a), they thus proposed the DAC-Att in a way that each embedding set independently computes attention scores and shares them with the other one to learn dual-aspect representations. Different from it, we propose a simple and generic mechanism by incorporating a multilayer perception (MLP). As shown in Fig. 4(b), besides the original self-attention scores (the orange squares), multiple auxiliary attention scores learned from other feature embeddings (the blue squares) are leveraged and fed into an element-wise MLP, which allows it to synthesize heterogeneous attention relationships into comprehensive ones. We call it ***Synthesis Attention (Synth-Att)***. It is able to not only leverage more attention scores from various feature embeddings, but also achieve competitive performance to DAC-Att with less computation costs. Below, we present more details.

Auxiliary Attention Scores. In our N2S, PFEs are used to generate multi-head auxiliary attention scores as follows,

$$\alpha^{\text{aux}}_{i,j,m} = \left(g_i W^{Q_{\text{aux}}}_m\right)\left(g_j W^{K_{\text{aux}}}_m\right)^T, \tag{6}$$

where $W^{Q_{\text{aux}}}_m \in \mathbb{R}^{d_g \times d_q}, W^{K_{\text{aux}}}_m \in \mathbb{R}^{d_g \times d_k}$ are trainable matrices for each head m. We set $m = 4$ and $d_q = d_k = d_g/m$.

Syn-Att. The Syn-Att is defined as follows,

$$\tilde{h}_i = \textbf{Syn-Att}(W^Q, W^K, W^V, W^O, \text{MLP}). \tag{7}$$

In specific, it first computes the multi-head self-attention scores $\alpha^{\text{self}}_{i,j,m}$ for NFEs based on the trainable matrices $W^Q_m \in \mathbb{R}^{d_h \times d_q}$ and $W^K_m \in \mathbb{R}^{d_h \times d_k}$ for head m using Eq. (8) as

$$\alpha^{\text{self}}_{i,j,m} = \left(h_i W^Q_m\right)\left(h_j W^K_m\right)^T. \tag{8}$$

Thereafter, the attention scores α^{aux} and α^{self} are fed into a three-layer MLP with structure $(2m \times 2m \times m)$ to compute the synthesized multi-head attention scores as follows,

$$\alpha^{\text{Synth}}_{i,j,1}, ..., \alpha^{\text{Synth}}_{i,j,m} = \text{MLP}\left(\alpha^{\text{self}}_{i,j,1}, ..., \alpha^{\text{self}}_{i,j,m}, \alpha^{\text{aux}}_{i,j,1}, ..., \alpha^{\text{aux}}_{i,j,m}\right). \tag{9}$$

The scores are then normalized to $\tilde{\alpha}_{i,j,m}$ through Softmax, which are further used to calculate the attention values for each head as Eq. (10). Finally, the outputs are given by Eq. (11) with trainable matrix $W^O \in \mathbb{R}^{md_v \times d_h}$ ($d_v = d_h/m$).

$$\text{head}_{i,m} = \sum_{j=1}^{|V|} \tilde{\alpha}_{i,j,m}\left(h_j W^V_m\right), \tag{10}$$

$$\tilde{h}_i = \text{Concat}\left[\text{head}_{i,1}, ..., \text{head}_{i,m}\right] W^O. \tag{11}$$

N2S Encoder. We stack L (L=3) encoders, each of which is the same as the Transformer encoder, except that the vanilla multi-head self-attention is replaced with our multi-head Synth-Att and we use the same instance normalization layer as [5]. Note that the auxiliary attention scores $\alpha^{\text{aux}}_{i,j,m}$ are only computed once and shared among all stacked encoders to reduce computation costs.

4.2.2 Decoder

The N2S decoder adopts the output from the encoder, which contains the embedding of the problem instance and the current solution, and generates action $a_t = \{(i^+, i^-), (j, k)\}$ as described in section 4.1.1 for removal and reinsertion. Thus the N2S decoder has two sub-decoders and each is responsible for deciding which node pair to remove (node-pair removal decoder) and where to reinsert (node-pair reinsertion decoder).

The N2S decoder first adopts the max-pooling layer in [8] to aggregate the global representation of all embeddings into each individual one as follows,

$$\hat{h}_i = \tilde{h}^{(L)}_i W^{Local}_h + \max\left[\{\tilde{h}^{(L)}_i\}^{|V|}_{i=1}\right] W^{Global}_h. \tag{12}$$

Node-Pair Removal Decoder. Given the enhanced embeddings $\{\hat{h}_i\}^{|V|}_{i=1}$ and the set $\mathcal{H}(t, K)$, the *removal* decoder outputs a categorical distribution over n requests for removal action. It first computes a *Node Closeness Score* λ_i (for each $x_i \in V$), indicating the closeness between node x_i and its neighbors as

$$\begin{aligned}\lambda_i = (\hat{h}_{\text{pred}(x_i)} W^Q_\lambda)(\hat{h}_i W^K_\lambda)^T + (\hat{h}_i W^Q_\lambda)(\hat{h}_{\text{succ}(x_i)} W^K_\lambda)^T \\ -(\hat{h}_{\text{pred}(x_i)} W^Q_\lambda)(\hat{h}_{\text{succ}(x_i)} W^K_\lambda)^T,\end{aligned} \tag{13}$$

where $\text{pred}(x_i)$ and $\text{succ}(x_i)$ refer to the first predecessor and the second successor nodes of x_i, respectively, and $W^Q_\lambda \in \mathbb{R}^{d_h \times d_h}$, $W^K_\lambda \in \mathbb{R}^{d_h \times d_h}$. The mixed use of the first and second neighbors is a balance between myopia (only first neighbors) and hyperopia (only second neighbors). We use multi-head technique to obtain $\lambda_{i,1}$ to $\lambda_{i,m}$. Then the decoder aggregates the scores for each pickup-delivery pair (i^+, i^-) based on a three-layer MLP_λ,

$$\begin{aligned}\tilde{\Lambda}_{(i^+,i^-)} = \text{MLP}_\lambda(\lambda_{i^+,1}, ..., \lambda_{i^+,m}, \lambda_{i^-,1}, ..., \lambda_{i^-,m}, \\ c(i), \mathbb{1}_{\text{last}(1)=i}, \mathbb{1}_{\text{last}(2)=i}, \mathbb{1}_{\text{last}(3)=i}),\end{aligned} \tag{14}$$

where the MLP structure is $(2m+4, 32, 32, 1)$, scalar $c(i)$ counts the frequency of request (i^+, i^-) being selected for removal in the past K steps, and $\mathbb{1}_{\text{last}(t)=i'}$ is a binary variable indicating whether request i' was selected at the t-th last step. Inspired by tabu search [32], we argue that taking into account the history of actions should be able to improve the decision making. The removal operation determines which pair of nodes to be removed, and the history information can effectively facilitate, e.g., by learning to avoid repetitive operations. An activation layer $\hat{\Lambda} = C \cdot \text{Tanh}(\tilde{\Lambda})$ is then applied ($C = 6$), followed by Softmax to normalize the distribution which is then used to sample a node pair (i^+, i^-) as the *removal* action.

Node-Pair Reinsertion Decoder. Given a request (i^+, i^-) for removal, the *reinsertion* decoder outputs the joint distribution that reinserts the two nodes back to the solution. We first define two *Node Reunite Scores* $\mu^{\text{p}}(x_\alpha, x_\beta)$ and $\mu^{\text{s}}(x_\alpha, x_\beta)$ for a node x_α, indicating the degree of preference of accepting a node x_β as its new predecessor and successor nodes, respectively,

$$\begin{aligned}\mu^{\text{p}}[x_\alpha, x_\beta] &= (\hat{h}_\alpha W^{Q_{\text{p}}}_\mu)(\hat{h}_\beta W^{K_{\text{p}}}_\mu)^T, \\ \mu^{\text{s}}[x_\alpha, x_\beta] &= (\hat{h}_\alpha W^{Q_{\text{s}}}_\mu)(\hat{h}_\beta W^{K_{\text{s}}}_\mu)^T,\end{aligned} \tag{15}$$

where $W^{Q_{\text{p}}}_\mu, W^{Q_{\text{s}}}_\mu \in \mathbb{R}^{d_h \times d_h}$, and $W^{K_{\text{p}}}_\mu, W^{K_{\text{s}}}_\mu \in \mathbb{R}^{d_h \times d_h}$. Again, we use multiple heads. Based on the scores, the decoder predicts the distribution of reinserting node i^+ after node j, and reinserting node i^- after node k using MLP_μ,

$$\begin{aligned}\tilde{\mu}[j, k] = \text{MLP}_\mu(\mu^{\text{p}}_1[i^+, \text{succ}(j)], ..., \mu^{\text{p}}_m[i^+, \text{succ}(j)], \\ \mu^{\text{p}}_1[i^-, \text{succ}(k)], ..., \mu^{\text{p}}_m[i^-, \text{succ}(k)], \\ \mu^{\text{s}}_1[i^+, j], ..., \mu^{\text{s}}_m[i^+, j], \mu^{\text{s}}_1[i^-, k], ..., \mu^{\text{s}}_m[i^-, k]),\end{aligned} \tag{16}$$

where the MLP structure is $(4m, 32, 32, 1)$. Note that here $\text{pred}(\cdot)$ and $\text{succ}(\cdot)$ should be considered in the new solution where nodes i^+, i^- have already been removed. Afterwards, $\hat{\mu} = C \cdot \text{Tanh}(\tilde{\mu})$ is applied and infeasible choices are masked as $-\infty$ before normalizing by Softmax. Finally, a node pair (j, k), as the *reinsertion* action, is sampled according to the resulting distribution to indicate the positions of reinserting the node-pair (i^-, i^+) back to the solution.

4.3 Policy Network of m-AM

Compared to the improvement policy in N2S, the network for the construction policy is less complicated since it mainly constructs a route as the initial solution to be improved by the N2S. In this sense, we leverage an encoder-decoder structured policy network for PDPs (as illustrated in Fig. 5) following the Attention Model (AM) [7]. The input to the construction policy includes all nodes in a PDP instance and the partially constructed solution, and it outputs a probability distribution for the action which is used to select the next node to visit. Note that our policy network is identical to the Heter-AM [9] except that the attention layer

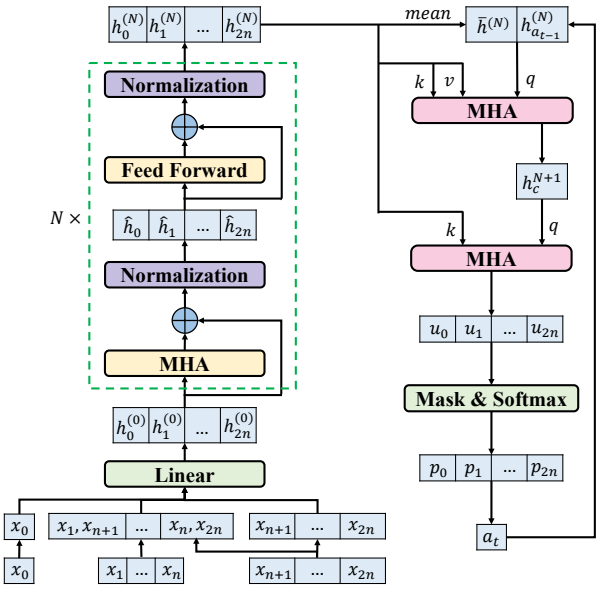


Fig. 5: m-AM policy network (left: encoder; right: decoder)

Algorithm 1 n-step PPO (shared-critic) with CL and IL strategy

Input: improvement policy π_θ, construction policy $\pi_{\theta'}$, critic v_ϕ, shared-critic parameter ζ, PPO clipping threshold ε, learning rate η_θ, $\eta_{\theta'}$, η_ϕ, η_ζ, learning rate decay β, epochs E, batches B, mini-batch κ, training steps T_{train}

for $e = 1$ to E **do**
  for $b = 1$ to B **do**
    Generate training data $\mathcal{D}_b$ on the fly;
    $\delta_0 \leftarrow$ **CurriculumLearning**$(\mathcal{D}_b, \pi_\theta, \pi_{\theta'}, e)$;
    Set initial state s_0 based on δ_0 and Eq. (1); $t \leftarrow 0$;
    while $t < T_{\text{train}}$ **do**
      $\mathcal{D}'_b \leftarrow$ **InvariantTransform**$(\mathcal{D}_b)$;
      Get $(\delta', f(\delta'))$ using $\pi_{\theta'}$ on $\mathcal{D}'_b$;
      Get $\{(s_\tau, a_\tau, r_\tau), f(\delta_\tau)\}_{\tau=t}^{t+n-1}$ using π_θ on $\mathcal{D}_b$;
      $t \leftarrow t + n$;
      Perform **PPO with shared-critic** to update θ, θ', ϕ, ζ;
    end while
    $\theta' \leftarrow$ **ImitationLearning**$(\pi_{\theta'}, \delta^*_{t-1}, \delta', \mathcal{D}_b, e)$
  end for
  $\eta_\theta \leftarrow \beta\eta_\theta$, $\eta_\phi \leftarrow \beta\eta_\phi$;
end for

in our encoder adopts the one in AM [7] which is simpler, so our m-AM can be considered as a lightweight Heter-AM. Three main differences are hold between m-AM and AM, including the input in the encoder, context embedding and mask scheme in the decoder. In this section, we will give a brief introduction to network design and highlight these three differences.

4.3.1 Encoder

The input to the encoder network is the coordinates of the $2n$+1 nodes in the Euclidean space. Given the property of the PDP, a representation enhancement is adopted, where each pickup node is concatenated with its paring delivery node [9], so the pickup node will be represented as a 4-dimension vector rather than 2-dimension. Three linear projection layers are used for the depot node, concatenated pickup nodes, and delivery nodes, respectively. Each linear projection layer encodes a 2-dimension (4-dimension for the concatenated pickup node) coordinate vector $l(x_i)$ to a d-dimension embedding $h_i^{(0)}$ with $d = 128$. This is where the first main difference lies.

Given the embeddings from the linear projection, they will be updated and promoted through N (N=3) attention layers. After passing through the N attention layers, the representation for each node will be transformed into a d-dimension embedding $h_i^{(N)}$. Then the graph embedding will be calculated as the mean of these node embeddings as

$$\bar{h}^{(N)} = \frac{1}{2n+1} \sum_{i=0}^{2n} h_i^{(N)}. \tag{17}$$

4.3.2 Decoder

Decoder aims to derive a probability distribution, based on which the action (or node) could be sequentially determined. It has three primary inputs, including 1) a $(2 * d)$-dimension context embedding $h_c^{(N)}$, which contains the graph embedding $\bar{h}^{(N)}$ as well as the embedding $h_{a_{t-1}}^{(N)}$ of the last node a_{t-1} visited at time step t, i.e., $h_c^{(N)} = \text{Concat}(\bar{h}^{(N)}, h_{a_{t-1}}^{(N)})$. Note that since a_0 is always the depot x_0, the real decoding will start at t=1; 2) the node embeddings $\{h_0^{(N)}, h_1^{(N)}, \ldots, h_{2n}^{(N)}\}$ from the encoder, which serves as the key and value for the attention computation in decoder; 3) a $(2n + 1)$-dimension mask vector, which ensures no duplicate nodes in a complete route and no violation of the precedence constraint of the pickup and delivery node pairs. As the context embedding and mask vector vary at each construction step, the partially constructed route is captured by masking the node from the context embedding at different steps. Note that the first and third inputs are related to the remaining two main differences.

These inputs will pass through two attention layers and then the final probability for action selection is calculated. According to the yielded probability vector, we sample the next node to visit. If the route has not yet been completely constructed, the sampled node will be used to update the context embedding $h_c^{(N)}$ and mask vector, based on which a new node to be visited next will be determined successively. This process is repeated until a complete route is constructed.

4.4 Training Algorithm

The overall architecture of our NCS approach is illustrated in Fig. 6, while the training flow is summarized in Algorithm 1, which is essentially a collaborative proximal policy optimization (PPO) [33] using the proposed shared-critic mechanism with curriculum learning (CL) and imitation learning (IL). It jointly learns an improvement policy π_θ (N2S) and a construction policy $\pi_{\theta'}$ (m-AM) with the help of a critic network v_ϕ for N2S, where the critic value for m-AM is attained based on v_ϕ.

Critic Network. Given the embeddings $\{\tilde{h}_i^{(L)}\}_{i=0}^{|V|}$ from N2S policy network π_θ, the critic network first enhances them by a vanilla multi-head attention layer (with $m = 4$ heads) to get $\{y_i\}_{i=0}^{|V|}$. The enhanced embeddings are then fed into a mean-pooling layer [8] to aggregate the global representation of all embeddings into each individual one as

$$\hat{y}_i = y_i W_v^{Local} + \text{mean}\left[\{y_i\}_{i=1}^{|V|}\right] W_v^{Global}, \tag{18}$$

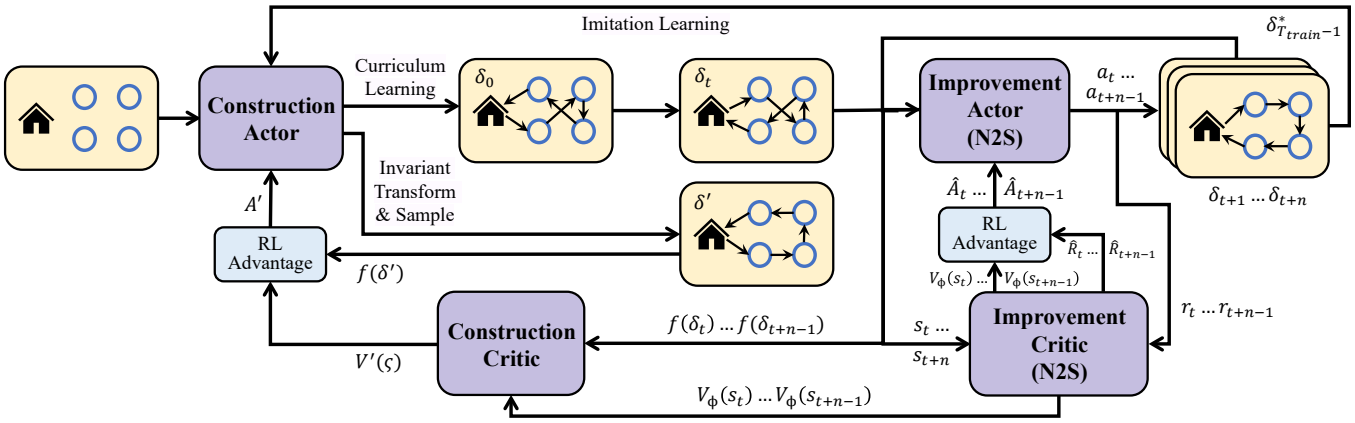


Fig. 6: The illustration of NCS training algorithm which exhibits the computation flow in a batch, all symbols are aligned with Algorithm 1.

where we use trainable matrices $W_v^{Local}, W_v^{Global} \in \mathbb{R}^{d_h \times \frac{d_h}{2}}$. Lastly, the state value $v(s_t)$ is output by a four-layer MLP in Eq. (19) with structure (129, 128, 64, 1). Here, $f(\delta^*)$ is added as an additional input to MLP_v.

$$v_\phi(s_t) = \text{MLP}_v\left(\max\left[\{\hat{y}_i\}_{i=1}^{|V|}\right], \text{mean}\left[\{\hat{y}_i\}_{i=1}^{|V|}\right], f(\delta^*)\right). \quad (19)$$

We train NCS for E epochs and B batches per epoch, where the training dataset $\mathcal{D}_b$ is randomly generated on the fly with the uniform distribution (line 3). While the improvement policy performs the gradient update for multiple times on the same PDP instance, the construction policy only needs to perform the gradient update once. Therefore, each time the improvement policy samples an action on the same instance, the construction policy will sample a solution on an invariant transformation (line 7) of the same instance to strengthen its exploration capability. The procedure of the invariant transformation is summarized in Algorithm 3 of Appendix C. The backbone of the whole algorithm basically follows the PPO equipped with our shared-critic mechanism, the detailed procedure is given in Algorithm 6 of Appendix C and we introduce shared-critic mechanism next.

Shared-Critic Mechanism. The critic network for the improvement model (N2S) takes its state s_t as the input, and fits the loss function in Eq. (24). We deduce that this critic network is trained to output a good estimation of the cumulative expected future rewards from the given state s_t, which is also the gap of objective value (i.e., route length) between solution δ_t and the optimal one. It means that the optimal objective for instance $\{l(x)\}_{x\in V}$ approximately equals $f(\delta_t) - v_\phi(s_t)$. In practice, we may not need a rigorously precise estimation during training, so we introduce a trainable parameter ζ, and thus the estimated optimal objective of the instance could become $f(\delta_t) - \zeta v_\phi(s_t)$. This can be potentially used as the critic of the construction model (m-AM), termed as the shared-critic mechanism, since it is derived directly from the improvement model. In practice, we use the n-step $f(\delta_t)$ and $v_\phi(s_t)$ to estimate this value, also known as the baseline of the construction model, i.e., $V'(\zeta) \leftarrow \text{mean}(\{f(\delta_\tau) - \zeta v_\phi(s_\tau)\}_{\tau=t}^{t+n-1})$, thus the construction advantage could be calculated as $A' \leftarrow f(\delta') - V'(\zeta)$ (line 9, 10 in Algorithm 6). Naturally, this shared-critic mechanism acts as a bridge to enable collaborative training with one critic network for two models. Compared with the traditional critic (i.e., a separate concrete network) [34] and rollout baseline (i.e., a backup of policy network) [7], our shared-critic has only one trainable parameter for the construction model, so the computation cost is greatly reduced.

Curriculum Learning. Following [5] and [35], we leverage the curriculum learning (CL) strategy (lines 4) to ameliorate the training further. In specific, a solution δ_0 is yielded from the current construction policy $\pi_{\theta'}$ and then the improvement policy π_θ uses it to set its initial state s_0. The quality of solution δ_0 will be improved as the epoch e increases. In such a way, the hardness level of neighborhood search is gradually increased per epoch because initiating searches from higher-quality solutions can make obtaining further improvements (rewards) more challenging. This follows the rationale of curriculum learning, which progressively increases task difficulty, thereby strengthening the capability of the policy to find improvements (rewards) even from near-optimal solutions, eventually aiming to discover the optimal solution. Meanwhile, such a CL strategy also narrows the search focus of the policy towards more refined regions around near-optimal solutions rather than broad, coarse searches on low-quality solutions, which is unnecessary given that the construction model can efficiently produce high-quality solutions without extensive search efforts. The input of CL includes the training data $\mathcal{D}_b$, both policies π_θ and $\pi_{\theta'}$, and current epoch e. It will output a solution as the initial state. The detailed procedure of CL is shown in Algorithm 4 of Appendix C.

Imitation learning. Inspired by [36], an imitation learning (IL) strategy is applied at the end of each batch. Curriculum learning enables the construction model to help the improvement model during collaborative training, while imitation learning enables the other way around. After the complete training with a small batch, we could obtain the best incumbent solution δ_{t-1}^* which is first yielded using curriculum learning and then improved for T_{train} steps by the improvement model. For the best incumbent solution δ_{t-1}^*, i.e., $f(\delta_{t-1}^*) \leq f(\delta_0)$, its quality will get better and better as the training iterations go on. As the training process for both models converges, δ_0 itself will also be of good quality. Given that δ_{t-1}^* has been improved for T_{train} steps from δ_0, it should approach the optimal solution. Hence δ_{t-1}^* could be used as a positive imitation sample. Thus, the construction policy could further update itself by imitating this sample (line 13). In particular, the loss function for imitating learning is expressed as

Algorithm 2 NCS(-A) Inference

Input: Instance $\mathcal{I}$ with size $|V|$, policy π_θ and $\pi_{\theta'}$, sample times S, maximum step T

1: **if** augment enabled **then**
2: **for** $i = 1, ..., \lfloor \frac{1}{2}|V| \rfloor$ **do**
3: $\mathcal{I}_i \leftarrow$ **InvariantTransform**$(\mathcal{I})$;
4: **end for**
5: **end if**
6: Sample S solutions for all instances $\mathcal{I}_i$ in parallel with $\pi_{\theta'}$;
7: Keep the best solution among these S solutions as the initial solution δ_0;
8: Solve all instances $\mathcal{I}_i$ with π_θ in parallel for T steps;
9: **return** the best solution found among all $\mathcal{I}_i$;

follows,

$$J'_{IL} = -\frac{1}{|\mathcal{D}_b|} \sum_{\mathcal{D}_b} (\xi \cdot \pi_{\theta'}(\delta^* | s'_0)), \tag{20}$$

where ξ is a tunable parameter to regulate the importance of imitation for the construction model. The input of IL includes the construction policy $\pi_{\theta'}$, the best incumbent solution δ^*_{t-1}, initial constructed solution δ', training data $\mathcal{D}_b$ and current epoch e. It outputs an updated construction policy $\pi_{\theta'}$ after imitation. The procedure of IL is summarized in Algorithm 5 of Apendix C.

4.5 Diversity Enhancement during Inference

Regarding the inference, NCS first yields the initial solution by sampling with the construction policy, and then passes it to the improvement policy as the starting solution for iterative updates. In the end, the best solution yielded by the improvement policy is retrieved as the final output.

To be more resistant to local minima, we further equip our NCS with an augmentation-based inference scheme, which leads to NCS-A in Algorithm 2. Such idea was originally explored in [6] for a neural *construction* method, and we extend it to an *improvement* one. The rationale is that an instance $\mathcal{I}$ can be transformed into different ones for searching while reserving the same optimal solution, e.g., rotating all locations of nodes by $\pi/2$ radian. For an instance of size $|V|$, our NCS-A performs $\lfloor \frac{1}{2}|V| \rfloor$ augments, each of which is generated by invariant transformation (Algorithm 3 of Appendix C). Note that although the mentioned transformations are conducted on instances defined in the Euclidean space, we believe that such an idea has favourable potential to be also exploited in non-Euclidean space, as long as there are proper invariant transformations for coordinates in the target space. Meanwhile, we use $K = |V|$ for training and $K = \lfloor \frac{1}{2}|V| \rfloor$ for inference. This is because we found that when a specific K in $\mathcal{H}(t, K)$ is used for training, a smaller K during inference can improve the diversity of the solutions, thus better eventual performance (Section 5.3).

4.6 Standalone N2S Approach

Removing the construction policy $\pi_{\theta'}$ from NCS, yields the standalone N2S approach [10], i.e., the improvement policy π_θ and critic network v_ϕ. This configuration serves as an independent neural improvement method for PDPs, facilitating a focused evaluation of the performance of the N2S policy network and supporting various experiments detailed in section 5. Meanwhile, since the construction model is removed, the IL strategy and shared-critic mechanism will not hold, and the CL strategy is degenerated as follows: it improves the randomly generated solution δ_{-1} to δ_0 by running the current policy π_θ for $T = e/\rho$ steps (e is current epoch number and ρ is a fixed hyperparameter) as proposed in [5], which means that we exploit the evolved policy itself to yield better and better initial solution as the training progresses. The improved solution δ_0 with higher quality (thus harder to improve) is used to initialize the first state s_0. For inference, the N2S approach will start from a randomly generated solution δ_0.

5 Evaluation

We design experiments to answer the following questions:

1) How good are the proposed NCS and N2S against the baselines, including the state-of-the-art neural methods and the strong LKH3 solver? (see Table 3 and Table 4)
2) Can N2S Synth-Att reduce computation costs while achieving competitive performance to DAC-Att and how crucial are the proposed learnable node-pair removal and node-pair reinsertion decoders for achieving an efficient search in N2S? (see Table 5 and Table 6)
3) How does the curriculum learning in NCS help the improvement model compared to the one trained individually and how does the imitation learning in NCS help the construction model? (see Table 9, Fig. 7 and Fig. 8)
4) How do variations in designs such as the action history length K and the MLP layer number in Synth-Att affect the performance of the N2S network? (see Table 7 and 8)
5) Can our NCS and N2S generalize well to benchmark instances that are different from training ones? (see Table 10)

5.1 Setup

We evaluate N2S and NCS on PDTSP and PDTSP-LIFO with three sizes $|V| = 21, 51, 101$ following the conventions in [5], [6], where the node coordinates of instances are randomly and uniformly generated in the unit square $[0, 1] \times [0, 1]$. For N2S, the initial solution δ_0 is sequentially constructed in a random fashion. Our experiments were conducted on a server equipped with 8 RTX 2080 Ti GPU cards and Intel E5-2680 CPU @ 2.4GHz. The training time of N2S varies with problem sizes, i.e., around 1 day for $|V| = 21$, 3 days for $|V| = 51$, and 7 days for $|V| = 101$. As for NCS, it is 1 day, 7 days and 10 days, respectively, both of which are shorter than all the neural baselines in Table 2. Regarding the memory cost, on PDTSP instance with $|V| = 51$, NCS training memory cost is 19918MB and N2S is 14786MB, the inference memory cost of NCS is 6136MB and N2S is 5758MB. We can see that NCS occupies about 34% more memory than N2S during training and only 6.5% more memory than N2S during inference. Our code is publicly available online[5].

Hyper-parameters. Our N2S and NCS are trained with $E = 200$ epochs and $B = 20$ batches per epoch using batch size 600. We set $n = 5$, $T_{\text{train}} = 250$ for the n-step PPO with $\kappa = 3$ mini-batch updates and a clip threshold $\epsilon = 0.1$. Adam optimizer is used with learning rate $\eta_\theta = 8 \times 10^{-5}$ for π_θ, $\eta_\phi = 2 \times 10^{-5}$ for v_ϕ (decayed $\beta = 0.985$ per epoch), and $\eta'_{\theta'} = 10^{-4}$ for $\pi'_{\theta'}$, $\eta_\zeta = 0.01$, both without decay. The reward discount factor γ is set to 0.999

5. https://github.com/dtkon/PDP-NCS.
6. https://github.com/Demon0312/Heterogeneous-Attentions-PDP-DRL
7. https://github.com/yd-kwon/POMO
8. https://github.com/yining043/VRP-DACT

TABLE 2: Training details of the adopted neural baselines.

Method	Code	Training Time	Hyper-parameters
Heter-AM	online[6]	$\sim$ 20 days	train 800 epochs as per the original setting.
Heter-POMO	online[7]	$\sim$ 14 days	train 2,000 epochs as per the original setting.
DACT	online[8]	$\sim$ 10 days	$\xi^{CL} = 0.25, 1, 4$ for sizes $\|v\|$ = 21, 51, 101, respectively; $n = 5$, $T_{\text{train}} = 250$ (same as ours); train 200 epochs (same as ours).

for both PDPs. We clip the gradient norm of N2S network to be within 0.05, 0.15, 0.3, and set the curriculum learning ρ to 2, 1.5, 1 for the three problem sizes, respectively. As for the construction model, its gradient norm is clipped to 1. Although ξ in Eq. (20) can regulate the strength of imitation learning, we actually use gradient clip to control this. We set $\xi = 1$ and clip the gradient norm of imitation learning to 0.1, 0.1, 0.01 for the three problem sizes. Particularly, the used hyperparameters regarding curriculum learning and imitation learning in NCS are shown in Table 12 of Appendix C.

5.2 Comparison Evaluation

We compare our N2S and NCS with the state-of-the-art (SOTA) neural methods and the highly-optimized LKH3 solver.

Regarding the former baseline, we consider the SOTA *improvement* method DACT [5][9] and the SOTA *construction* method Heter-AM [9] (specially designed for PDPs). To make a fair comparison with our N2S-A and NCS-A (with the diversity enhancement), we upgrade Heter-AM to Heter-POMO, also given the known superiority of POMO to AM. In specific, we reserve the policy network in Heter-AM as the backbone while adopting the diverse rollouts and the data augmentation techniques in POMO [6] to leverage the advantages of them for the best performance. Each neural baseline is trained using the respective implementation code that is publicly available. For the upgraded Heter-POMO, we adapt and combine the model architecture from the original Heter-AM and the original POMO. The links to their original implementations, approximate training time for the size $|V| = 101$, and the used hyper-parameters are presented in Table 2. For other hyper-parameters, we follow the recommendations in their papers. Regarding the latter baseline, LKH3 is a strong heuristic (as reviewed in Section 2) which is widely used as a baseline to benchmark neural methods in recent studies (e.g., [5], [8], [21], [37]). We report its results with two settings of iterations, i.e., LKH (5k) and LKH (10k).

All baselines are evaluated on a test dataset with 2,000 instances, and we report the metrics of averaged objective values, standard deviation of objective values, averaged gaps to LKH (10K) and the total solving time in Table 3 and 4. The averaged gap is equal to $(\text{Obj} - \text{Obj}_{\text{LKH}})/\text{Obj}_{\text{LKH}} \times 100$, where Obj is the average objective value of the target method and Obj_{LKH} is the average objective value of LKH (10k). In the above two tables, bold gaps indicate that our methods significantly outperform the best neural method (i.e., Heter-POMO-A), and underlined gaps indicate that our methods significantly outperform LKH (10k), according to a paired t-test (α = 5%), on 2,000 test instances. The detailed results of t-tests are shown in Table 13 of Appendix D.

9. We use the *insert* decoder which is the best for PDPs.

Note that it is hard to perform an absolutely fair time comparison between running Python codes on GPUs (neural methods) and running ANSI C codes on CPUs (LKH solver). Thus we follow the guidelines in [38] to perform the *facilitate comparison* that lets each method make full use of the best settings on our machine. In particular, we report the time of LKH3 when running in parallel with 16 CPU cores and the time of each neural method when all 8 GPU cards are available (but do not need to be fully used).

Results on PDTSP. Table 3 shows the results on PDTSP. In the first group, we compare N2S and NCS with Heter-AM (*greedy* and *sampling*), and DACT. Compared to Heter-AM (5k), our N2S with only 1k steps attains lower gaps with less time for all sizes. Although DACT offers the best gap on PDTSP-21, its performance drops significantly as the problem size increases, partly because its decoder is less efficient than our node-pair *removal* and *reinsertion* ones when tackling larger-scale problems. Instead, our N2S achieves higher performance and consistently dominates DACT in terms of both the gaps and the time on PDTSP-51 and PDTSP-101. As for NCS, the performance is further boosted compared to N2S, with better results for all three different total steps. Even NCS with 1k steps has surpassed N2S with 3k steps, without incurring much extra consumption time. In the second group, our augmented N2S-A and NCS-A is compared to the upgraded Heter-POMO method with three variants[10]. It is shown that even with only 1k steps, our N2S-A attains a significantly smaller gap than all three Heter-POMO variants, by almost an order of magnitude. Although Heter-POMO-A (gr.) tends to be competitive with fast speed, the gap is hard to be further reduced by increasing inference time if we refer to Heter-POMO-A (3k). Moreover, our N2S-A (2k) keeps abreast of, or even slightly exceeds the strong LKH3 solver, achieving gaps of -0.03% and -0.01% with less time on PDTSP-51 and PDTSP-101, respectively. Those gaps are further reduced to -0.04% and -0.20% with more steps, i.e., 3k. And NCS-A can further extend the advantage to -0.06% and -0.22%. Although our N2S-A and NCS-A fail to significantly outperform LKH3 at 2k steps, they have lower standard deviations, implying more stable solving power. With more run time, both our N2S-A and NCS-A significantly exceed LKH3 on PDTSP-101 with 3k steps. Notice that we observe that there is no significant difference between N2S-A and NCS-A, which implies that the results of the two approaches may be fairly close to the real optimum, with the help of diversity enhancement.

Results on PDTSP-LIFO. In Table 4, we report the results on PDTSP-LIFO. Due to the more constrained search space, DACT failed to work well (see Table 6). Therefore, we mainly compare our approaches with Heter-POMO (the best neural baseline in Table 3) and the LKH3 solver. As exhibited, the advantages of neural methods over LKH3 have been further enhanced on this harder problem, where our approach consistently outperforms Heter-POMO for all sizes. Compared to LKH3, our N2S-A with 3k steps presents superior performance again, and attains gaps of -0.64% and -1.47% on PDTSP-LIFO-51 and PDTSP-LIFO-101, respectively. And our NCS-A achieves even better results than that of N2S-A, with gaps of -0.65% and -1.68%, respectively. The advantages of our approaches on PDTSP-LIFO are more obvious, both N2S-A and NCS-A significantly outperform LKH3 with 1k steps on PDTSP-LIFO-51 and PDTSP-LIFO-101.

10. Heter-POMO (gr.), Heter-POMO-A (gr.), Heter-POMO-A (3k) refer to: greedily generate $|V|$ solution; greedily generate $|V|$ solutions with 8 augments; sample $|V| \times$3k solutions with 8 augments.

TABLE 3: Results for PDTSP with sizes $|V| = 21, 51, 101$. The "+" in "Total Time" means construction time plus improvement time.

Methods	PDTSP-21			PDTSP-51			PDTSP-101		
	Obj. Value ↓ (±SD)	Gap to LKH ↓	Total Time	Obj. Value ↓ (±SD)	Gap to LKH ↓	Total Time	Obj. Value ↓ (±SD)	Gap to LKH ↓	Total Time
LKH (5k)	4.563 ± 0.3806	0.00%	3m	6.866 ± 0.3151	0.06%	10m	9.443 ± 0.3010	0.16%	49m
LKH (10k)	4.563 ± 0.3806	0.00%	5m	6.862 ± 0.3132	0.00%	19m	9.428 ± 0.2971	0.00%	98m
Heter-AM (gr.)	4.655 ± 0.4128	2.02%	(0s)	7.333 ± 0.4113	6.86%	(1s)	10.348 ± 0.4127	9.76%	(2s)
Heter-AM (5k)	4.578 ± 0.3877	0.33%	(33s)	7.108 ± 0.3591	3.58%	(1.5m)	10.051 ± 0.3644	6.61%	(5m)
DACT (1k)	4.572 ± 0.3866	0.20%	(18s)	7.245 ± 0.4077	5.57%	(29s)	10.551 ± 0.4877	11.91%	(51s)
DACT (2k)	4.566 ± 0.3823	0.07%	(37s)	7.118 ± 0.3834	3.72%	(1m)	10.312 ± 0.4480	9.38%	(1.5m)
DACT (3k)	4.564 ± 0.3817	0.03%	(1m)	7.057 ± 0.3718	2.83%	(1.5m)	10.195 ± 0.4317	8.13%	(2.5m)
N2S (1k)	4.573 ± 0.3842	0.21%	(21s)	7.103 ± 0.3791	3.51%	(31s)	10.030 ± 0.4001	6.38%	(1m)
N2S (2k)	4.567 ± 0.3823	0.09%	(42s)	7.053 ± 0.3684	2.77%	(1m)	9.905 ± 0.3784	5.06%	(2m)
N2S (3k)	4.565 ± 0.3817	0.05%	(1m)	7.027 ± 0.3621	2.40%	(1.5m)	9.846 ± 0.3663	4.44%	(3m)
NCS (0)	4.654 ± 0.4159	1.99%	(1s)	7.422 ± 0.3968	8.16%	(7s)	11.039 ± 0.3811	17.09%	(46s)
NCS (1k)	4.570 ± 0.3833	0.15%	(1s+21s)	6.974 ± 0.3471	1.63%	(7s+31s)	9.808 ± 0.3537	4.03%	(46s+1m)
NCS (2k)	4.567 ± 0.3818	0.09%	(1s+42s)	6.957 ± 0.3415	1.38%	(7s+1m)	9.757 ± 0.3456	3.49%	(46s+2m)
NCS (3k)	4.565 ± 0.3814	0.05%	(1s+1m)	6.948 ± 0.3394	1.25%	(7s+1.5m)	9.730 ± 0.3395	3.20%	(46s+3m)
Heter-POMO (gr.)	4.634 ± 0.4046	1.56%	(0s)	7.168 ± 0.3763	4.45%	(1s)	10.060 ± 0.3739	6.70%	(2s)
Heter-POMO-A (gr.)	4.584 ± 0.3886	0.46%	(1s)	6.995 ± 0.3331	1.93%	(5s)	9.681 ± 0.3013	2.68%	(11s)
Heter-POMO-A (3k)	4.564 ± 0.3822	0.03%	(7m)	6.916 ± 0.3215	0.77%	(32m)	9.567 ± 0.2891	1.47%	(135m)
N2S-A (1k)	4.563 ± 0.3807	0.01%	(1m)	6.865 ± 0.3130	**0.03**%	(8m)	9.475 ± 0.2816	**0.50**%	(40m)
N2S-A (2k)	4.563 ± 0.3807	0.00%	(2m)	6.860 ± 0.3116	**-0.03**%	(16m)	9.427 ± 0.2783	**-0.01**%	(80m)
N2S-A (3k)	4.563 ± 0.3807	0.00%	(3m)	6.860 ± 0.3110	**-0.04**%	(24m)	9.409 ± 0.2770	**-0.20**%	(121m)
NCS-A (0)	4.595 ± 0.3931	0.70%	(6s)	7.192 ± 0.3618	4.81%	(35s)	10.640 ± 0.3461	12.86%	(9m)
NCS-A (1k)	4.563 ± 0.3806	0.00%	(6s+1m)	6.864 ± 0.3129	**0.03**%	(35s+8m)	9.471 ± 0.2811	**0.46**%	(9m+40m)
NCS-A (2k)	4.563 ± 0.3806	0.00%	(6s+2m)	6.860 ± 0.3114	**-0.03**%	(35s+16m)	9.424 ± 0.2791	**-0.04**%	(9m+80m)
NCS-A (3k)	4.563 ± 0.3806	0.00%	(6s+3m)	6.858 ± 0.3107	**-0.06**%	(35s+24m)	9.407 ± 0.2749	**-0.22**%	(9m+121m)

TABLE 4: Results for PDTSP-LIFO with sizes $|V| = 21, 51, 101$. The "+" in "Total Time" means construction time plus improvement time.

Methods	PDTSP-LIFO-21			PDTSP-LIFO-51			PDTSP-LIFO-101		
	Obj. Value ↓ (±SD)	Gap to LKH ↓	Total Time	Obj. Value ↓ (±SD)	Gap to LKH ↓	Total Time	Obj. Value ↓ (±SD)	Gap to LKH ↓	Total Time
LKH (5k)	5.539 ± 0.5255	0.00%	(1m)	10.218 ± 0.6047	0.17%	(8m)	17.115 ± 0.7573	0.34%	(33m)
LKH (10k)	5.539 ± 0.5255	0.00%	(3m)	10.200 ± 0.5995	0.00%	(16m)	17.057 ± 0.7406	0.00%	(67m)
N2S (1k)	5.541 ± 0.5261	0.04%	(2m)	10.365 ± 0.6189	1.61%	(2.5m)	17.660 ± 0.7594	3.54%	(3.5m)
N2S (2k)	5.540 ± 0.5257	0.02%	(4m)	10.301 ± 0.6065	0.99%	(5.5m)	17.471 ± 0.7364	2.43%	(6.5m)
N2S (3k)	5.539 ± 0.5256	0.01%	(6m)	10.265 ± 0.6009	0.64%	(8m)	17.373 ± 0.7256	1.85%	(10m)
NCS (0)	5.674 ± 0.5578	2.44%	(1s)	10.993 ± 0.6703	7.77%	(7s)	19.274 ± 0.7967	13.00%	(46s)
NCS (1k)	5.540 ± 0.5262	0.02%	(1s+2m)	10.299 ± 0.6071	0.97%	(7s+2.5m)	17.466 ± 0.7618	2.40%	(46s+3.5m)
NCS (2k)	5.539 ± 0.5257	0.01%	(1s+4m)	10.260 ± 0.6004	0.59%	(7s+5.5m)	17.343 ± 0.7457	1.68%	(46s+6.5m)
NCS (3k)	5.539 ± 0.5257	0.01%	(1s+6m)	10.242 ± 0.5973	0.41%	(7s+8m)	17.276 ± 0.7422	1.29%	(46s+10m)
Heter-POMO (gr.)	5.636 ± 0.5477	1.75%	(0s)	10.540 ± 0.6429	3.33%	(1s)	17.583 ± 0.7537	3.08%	(2s)
Heter-POMO-A (gr.)	5.567 ± 0.5310	0.51%	(1s)	10.353 ± 0.6090	1.50%	(5s)	17.276 ± 0.7210	1.29%	(10s)
Heter-POMO-A (4k)	5.545 ± 0.5278	0.12%	(10m)	10.209 ± 0.5908	0.09%	(48m)	16.890 ± 0.6908	-0.98%	(180m)
N2S-A (1k)	5.539 ± 0.5256	0.00%	(3m)	10.144 ± 0.5822	**-0.55**%	(13m)	16.976 ± 0.7019	-0.47%	(54m)
N2S-A (2k)	5.539 ± 0.5256	0.00%	(6m)	10.137 ± 0.5805	**-0.62**%	(27m)	16.859 ± 0.6955	**-1.16**%	(107m)
N2S-A (3k)	5.539 ± 0.5256	0.00%	(9m)	10.135 ± 0.5800	**-0.64**%	(40m)	16.806 ± 0.6934	**-1.47**%	(161m)
NCS-A (0)	5.588 ± 0.5385	0.88%	(6s)	10.644 ± 0.6353	4.35%	(35s)	18.672 ± 0.7695	9.47%	(9m)
NCS-A (1k)	5.539 ± 0.5255	0.00%	(6s+3m)	10.142 ± 0.5821	**-0.57**%	(35s+13m)	16.894 ± 0.7090	-0.96%	(9m+54m)
NCS-A (2k)	5.539 ± 0.5255	0.00%	(6s+6m)	10.136 ± 0.5808	**-0.63**%	(35s+27m)	16.815 ± 0.7027	**-1.42**%	(9m+107m)
NCS-A (3k)	5.539 ± 0.5255	0.00%	(6s+9m)	10.134 ± 0.5801	**-0.65**%	(35s+40m)	16.771 ± 0.6965	**-1.68**%	(9m+161m)

The NCS (0) and NCS-A (0) in Table 3 and 4 represent the initial objective value for starting the improvement, that is, the standalone performance of the construction model (m-AM). Compared with Heter-AM, such performance is reasonable, considering that m-AM is a simplified version of Heter-AM with less intensive training (Heter-AM undergoes 800 epochs with 1,280,000 instances per epoch, whereas m-AM in NCS trains for only 200 epochs with 12,000 instances per epoch). Note that we are not trying to train an m-AM to work independently, but to train an m-AM to get the best results from cooperation with N2S in NCS, so we focus more on NCS (3k) rather than NCS (0). In subsequent ablation studies, we will highlight the key role of collaborative training in enhancing cooperative performance compared to when models are trained separately and then combined.

5.3 Ablation Evaluation

Effects of Different Encoding Methods in N2S. We replace the proposed Synth-Att in our N2S encoder with vanilla-Att and DAC-Att, respectively. Using only one GPU card, we report the number of model parameters, time, and gaps for solving 2,000 PDTSP-51 instances with 3k steps in Table 5. As exhibited, Synth-Att attains slightly smaller gaps than DAC-Att with much fewer computation costs, and considerably lower gaps than vanilla-Att with slight extra computation costs.

TABLE 5: Effects of different encoding methods.

Att. in Encoders	Dim.	# Param.(M)	Time(s)	Gap(%) ↓
Vanilla-Att	64	0.18 (1.00×)	239 (1.00×)	4.66
DAC-Att		0.31 (1.72×)	285 (1.19×)	2.89
Synth-Att		**0.19 (1.06×)**	**255 (1.07×)**	**2.88**
Vanilla-Att	128	0.72 (1.00×)	322 (1.00×)	3.92
DAC-Att		1.25 (1.73×)	400 (1.24×)	2.43
Synth-Att		**0.76 (1.06×)**	**340 (1.06×)**	**2.40**

Effects of Different Decoding Methods in N2S. To highlight the desirability of our two decoders, we replace the trainable decoders with hand-crafted ones (i.e., random and the ϵ-greedy with $\epsilon = 0.1$) while ensuring feasibility. We gather the results on PDTSP-51 and PDTSP-LIFO-51 with 3k steps in Table 6 where mark '✓' means our proposed decoder is used (the one without augments) and mark '✗' means the decoder in the parentheses is used instead. As revealed, the methods of retaining at least one trainable decoder always attain much lower gaps than the ones equipped with only

TABLE 6: Effects of different decoding methods.

Removal Decoder	Reinsertion Decoder	Gap(%) on PDTSP ↓	Gap(%) on PDTSP-LIFO ↓
✗(random)	✗(random)	210.82	112.37
✗(random)	✗(ϵ-greedy)	18.03	17.87
✗(ϵ-greedy)	✗(random)	86.31	47.57
✗(ϵ-greedy)	✗(ϵ-greedy)	15.62	12.64
✓	✗(random)	43.12	17.75
✓	✗(ϵ-greedy)	3.54	3.88
✗(random)	✓	6.38	4.38
✗(ϵ-greedy)	✓	8.42	6.28
✗(DACT)	✗(DACT)	2.83	11.73
✓	✓	**2.40**	**0.64**

TABLE 7: Effects of different action history length.

K (training)	K (inference)							
	Obj. Value ↓				Gap to LKH ↓			
	0	25	50	100	0	25	50	100
0	7.087	7.091	7.091	7.09	3.28%	3.34%	3.34%	3.32%
50	7.111	**7.027**	7.036	7.054	3.63%	**2.40%**	2.54%	2.80%
100	7.102	7.035	7.038	7.043	3.50%	2.52%	2.56%	2.64%

hand-crafted decoders. The method with both trainable decoders (i.e., N2S) achieves the best performance. We also notice that DACT fails to solve PDTSP-LIFO well. This might be because its decoder considers removing and reinserting only one node instead of a pickup-delivery node pair in each action, which is a serious limitation for more constrained PDPs. For example, on PDTSP-LIFO, over 92% of the action space needs to be masked for DACT, which leads to extremely low efficiency.

Effects of Different Action History Length in N2S. In section 4.1.1, we mentioned that the state to N2S includes an action history $\mathcal{H}(t, K)$ that stores the most recent K actions at time step t. And in section 4.5 we recommend to use $K = |V|$ for training and $K = \lfloor \frac{1}{2}|V| \rfloor$ for inference. In order to justify our suggestion, we set different K for training and inference on PDTSP-51, and observe their experimental results of N2S with 3k steps in Table 7. It can be seen that when $K = 50$ in training, the best result is obtained by using $K = 25$ in inference. And if $K = 100$ for training, $K = 50$ or 25 for inference can get relatively good results. Training with $K = 0$ does not incur the ability to gain experience from historical steps, so using any K during inference gives similar results. It can be found that when a specific K is used for training, a smaller K during inference can improve the performance. This might be due to that a smaller K during inference corresponds to a lower degree of confidence in the prediction of the model, which in turn can increase the degree of exploration of the action, thus enhancing the diversity of solutions and eventually better inference performance.

Effects of MLP layer number in N2S Syn-Att. In Eq. (9) of section 4.2.1, we adopt a three-layer MLP with structure ($2m \times$

TABLE 8: Effects of MLP layer number.

Structure	Step					
	Obj. Value ↓			Gap to LKH ↓		
	1k	2k	3k	1k	2k	3k
MLP-2	7.126	7.072	7.052	3.85%	3.06%	2.77%
MLP-3	7.103	7.053	**7.027**	3.51%	2.78%	**2.40%**
MLP-4	7.106	7.056	7.034	3.56%	2.83%	2.51%

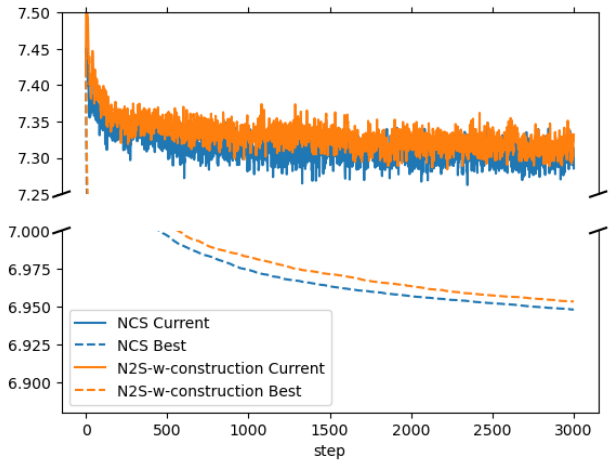


Fig. 7: Convergence curve of NCS, and N2S with Construction

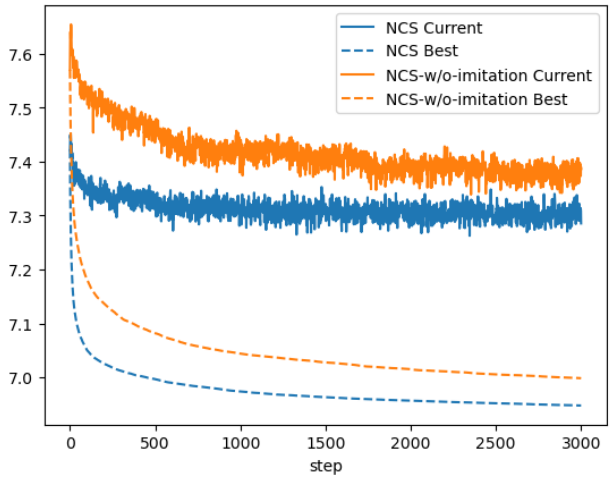
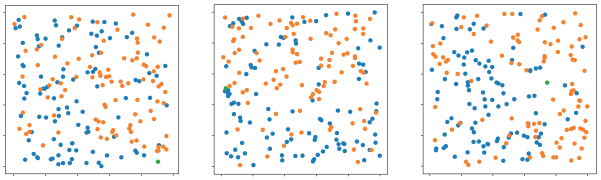


Fig. 8: Convergence curve of NCS, and NCS without IL

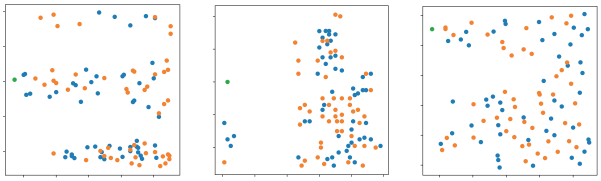
$2m \times m$) to compute the synthesized multi-head attention scores. In Table 8, we present experimental results of N2S using different numbers of MLP layer on PDTSP-51. MLP-2 is a two-layer MLP with structure ($2m \times m$) and MLP-4 is a four-layer MLP with structure ($2m \times 2m \times 2m \times m$). It can be seen that the difference between the three structures is very small. Among them, MLP-2 yields the worst performance due to its simplistic architecture. Conversely, the performance difference between MLP-3 and MLP-4 is negligible. Overall, the variation in the number of MLP layers exerts minimal influence on the effectiveness of N2S. Given that MLP-3 achieves the best balance of complexity and performance, we choose it in our N2S implementation.

Effects of Curriculum Learning in NCS. To assess the effects of the CL in NCS, we compare the improvement policy trained collaboratively with the construction policy using NCS (with CL) and the one trained independently using N2S (with degenerated CL). Meanwhile, we use the construction policy trained in NCS to also generate initial solutions to boost the latter improvement model during the inference for a fair comparison, named N2S-w-construction. The comparison results on PDTSP-51 instances, with and without the diversity enhancement, are gathered in Table 9. As can be seen, our NCS consistently outperforms N2S-w-construction regardless of the diversity enhancement. In Fig. 7,



(a) N201P3 (b) N201P6 (c) N201P7

Fig. 9: Example instances from [29].



(a) brd14051_101 (b) pr1002_101 (c) fnl4461_101

Fig. 10: Example instances from [31].

TABLE 9: Effects of curriculum learning and imitation learning.

Method	Step					
	Obj. Value ↓			Gap to LKH ↓		
	1k	2k	3k	1k	2k	3k
NCS	6.974	6.957	**6.948**	1.63%	1.38%	**1.25%**
N2S-w-construction	6.982	6.963	6.953	1.75%	1.47%	1.33%
NCS-w/o-imitation	7.043	7.012	6.996	2.64%	2.19%	1.95%
NCS-A	6.864	6.860	**6.858**	0.03%	-0.03%	**-0.06%**
N2S-A-w-construction	6.865	6.861	6.860	0.04%	-0.01%	-0.03%
NCS-A-w/o-imitation	6.866	6.861	6.859	0.06%	-0.01%	-0.04%

TABLE 10: Generalization performance on benchmark instances.

Problem	$\|V\|$	Gaps to	Heter-POMO-A		N2S-A		NCS-A	
			Avg. ↓	Best ↓	Avg. ↓	Best ↓	Avg. ↓	Best ↓
PDTSP	101	Opt.	1.64%	1.46%	0.08%	**0.00%**	**0.03%**	**0.00%**
	201		9.71%	8.66%	2.82%	2.19%	**2.24%**	**1.93%**
PDTSP -LIFO	≤101	B1	9.63%	8.80%	0.81%	**-0.17%**	**0.42%**	-0.04%
		B2	10.98%	10.14%	2.02%	**1.02%**	**1.62%**	1.15%

we further depict the averaged search curves of the best-so-far and current solutions during inference for both methods, where we observe that although both improvement policies start with identical initial solutions, their iterative trajectories differ a lot. Our NCS explores regions of lower objective values earlier and converges to better solutions faster. This may be attributed to the distinct CL strategies that result in different scopes of the initial solutions during training for the improvement policies, where the initial solutions yielded by the construction model with the CL strategy often lead to more desirable areas that are closer to the optimal solutions.

Effects of Imitation Learning in NCS. As shown in Table 9, the absence of imitation learning in NCS would greatly affect the performance of the construction model and thus the eventual results. The construction model in NCS is lightweight and uses considerably fewer computational resources compared to individually learned models (e.g., Heter-AM samples 426.67 times more instances during training than our construction model). Consequently, relying solely on reinforcement learning (i.e., without imitation learning) leads to underfitting for the construction model, resulting in poor initial solutions as illustrated in Fig. 8. This hinders the effectiveness of collaborative training, as the scope of initial solutions provided by the construction model is not adequately desirable, thus undermining the overall performance.

5.4 Generalization Evaluation

We further evaluate our N2S and NCS on benchmark instances, including all the ones from [29] for PDTSP and the ones with size $|V| \leq 201$ from [31] for PDTSP-LIFO, which are largely different from our training ones, e.g., different sizes (i.e., 200 nodes) as shown in Fig. 9 and different node distributions as shown in Fig. 10. In Table 10, we report the best and the average gaps (with 10 independent runs) achieved by N2S-A, NCS-A and neural baseline Heter-POMO-A w.r.t. optimal solutions for PDTSP, or heuristic baseline B1 [31] and B2 [4] for PDTSP-LIFO. It can be seen that our N2S and NCS significantly outstrip Heter-POMO in all cases, with NCS further strengthening the advantage over N2S. Without re-training or leveraging other techniques, this is fairly hard to achieve because machine learning often suffers from a mediocre out-of-distribution zero-shot generalization [39], [40]. The results also imply slight inferiority of our N2S and NCS to the LKH3 solver, given that LHK3 reports similar performance to the B2 baseline on those instances [12]. Accordingly, we will focus on further improving the out-of-distribution generalization performance for our N2S and NCS in the future.

6 Conclusion and Discussion

We present the neural collaborative search (NCS) approach for PDPs, illustrating the substantial potential for collaboration between the latest prevalent neural construction and neural improvement models. Our NCS employs a construction model to augment the improvement model, integrating both within a unified reinforcement learning paradigm via a shared-critic mechanism. Meanwhile, they also enhance each other through upgraded curriculum learning and imitation learning during the collaborative training, respectively. On the other hand, the improvement model in NCS is an efficient neural neighborhood search (N2S) approach for PDPs. It utilizes a novel Synth-Att to synthesize node relationships from various types of solution features and exploits the node-pair removal and reinsertion decoders to tackle the precedence constraint. Extensive experiments on PDTSP and PDTSP-LIFO verified our design, where NCS and N2S achieve state-of-the-art performance among existing neural methods. Further equipped with a diversity enhancement scheme, they even become the first neural methods to surpass LKH3 on synthesized PDP instances.

Despite the promising results, we believe that it is still not the time to conclude that learned solvers like our NCS and N2S can fully replace traditional solvers like LKH3, especially for the out-of-distribution performance. Nevertheless, our proposed methods have exhibited distinct benefits: 1) Our NCS and N2S models, through a unified training framework, automatically learn to specialize in two PDP variants, which reduces the need for extensive problem-specific knowledge and time-consuming heuristic designs through trials and errors; 2) They could leverage GPU parallelism to concurrently solve thousands of PDP instances, desirable for industrial use; and 3) Our N2S and NCS are the first to surpass LKH3 on PDPs, which demonstrates that, in merely five years, neural methods including NCS and N2S have matched or even outperformed well-established hand-crafted solvers, e.g.,

LKH3, which have evolved for several decades, thereby showing significant rapid progress and potential for future advancements.

For future works, we will 1) deploy NCS or N2S as a low-level agent in the hierarchical framework of [41] for dynamic PDPs, 2) combine NCS or N2S with a similar divide-and-conquer strategy in [37] for much larger-scale instances, 3) enhance the out-of-distribution generalization for NCS and N2S so as to exceed LKH3 on instances of arbitrary distributions, 4) investigate different types of construction and improvement models to be trained collaboratively in NCS and solve more other VRP variants (e.g., employing the construction model POMO [6] and improvement model NeuOpt [23] within the NCS framework for TSP or CVRP).

ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China (62072258); the Special Foundation for Philosophy and Social Science Laboratories of Ministry of Education of China [grant number H0123702]; the Asia Research Center in Nankai University [grant number AS2405]. It was also supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG3-RP-2022-031), and the Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 1 grant.

REFERENCES

[1] S. N. Parragh, K. F. Doerner, and R. F. Hartl, "A survey on pickup and delivery problems," *Journal für Betriebswirtschaft*, vol. 58, no. 2, pp. 81–117, 2008.

[2] S. Ropke and D. Pisinger, "A unified heuristic for a large class of vehicle routing problems with backhauls," *European Journal of Operational Research*, vol. 171, no. 3, pp. 750–775, 2006.

[3] V. Ghilas, E. Demir, and T. Van Woensel, "An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows and scheduled lines," *Computers & Operations Research*, vol. 72, pp. 12–30, 2016.

[4] Y. Li, A. Lim, W.-C. Oon, H. Qin, and D. Tu, "The tree representation for the pickup and delivery traveling salesman problem with LIFO loading," *European Journal of Operational Research*, vol. 212, no. 3, pp. 482–496, 2011.

[5] Y. Ma, J. Li, Z. Cao, W. Song, L. Zhang, Z. Chen, and J. Tang, "Learning to iteratively solve routing problems with dual-aspect collaborative Transformer," in *Advances in Neural Information Processing Systems*, vol. 34, 2021, pp. 11 096–11 107.

[6] Y.-D. Kwon, J. Choo, B. Kim, I. Yoon, Y. Gwon, and S. Min, "POMO: Policy optimization with multiple optima for reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 21 188–21 198.

[7] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!" in *International Conference on Learning Representations*, 2018.

[8] Y. Wu, W. Song, Z. Cao, J. Zhang, and A. Lim, "Learning improvement heuristics for solving routing problems," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[9] J. Li, L. Xin, Z. Cao, A. Lim, W. Song, and J. Zhang, "Heterogeneous attentions for solving pickup and delivery problem via deep reinforcement learning," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[10] Y. Ma, J. Li, Z. Cao, W. Song, H. Guo, Y. Gong, and Y. M. Chee, "Efficient neural neighborhood search for pickup and delivery problems," in *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*, 2022, pp. 4776–4784.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30, 2017, pp. 6000–6010.

[12] K. Helsgaun, "An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems," *Roskilde: Roskilde University*, 2017.

[13] M. Nazari, A. Oroojlooy, M. Takáč, and L. V. Snyder, "Reinforcement learning for solving the vehicle routing problem," in *Advances in Neural Information Processing Systems*, 2018, pp. 9861–9871.

[14] C. K. Joshi, T. Laurent, and X. Bresson, "An efficient graph convolutional network technique for the travelling salesman problem," arXiv preprint arXiv: 1906.01227, 2019.

[15] M. Kim, J. Park, and j. kim, "Learning collaborative policies to solve NP-hard routing problems," in *Advances in Neural Information Processing Systems*, vol. 34, 2021, pp. 10 418–10 430.

[16] Z.-H. Fu, K.-B. Qiu, and H. Zha, "Generalize a small pre-trained model to arbitrarily large TSP instances," in *AAAI Conference on Artificial Intelligence*, 2021.

[17] J. Chen, Z. Zhang, Z. Cao, Y. Wu, Y. Ma, T. Ye, and J. Wang, "Neural multi-objective combinatorial optimization with diversity enhancement," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[18] Q. Hou, J. Yang, Y. Su, X. Wang, and Y. Deng, "Generalize learned heuristics to solve large-scale vehicle routing problems in real-time," in *The Eleventh International Conference on Learning Representations*, 2023.

[19] X. Chen and Y. Tian, "Learning to perform local rewriting for combinatorial optimization," in *Advances in Neural Information Processing Systems*, vol. 32, 2019, pp. 6281–6292.

[20] A. Hottung and K. Tierney, "Neural large neighborhood search for the capacitated vehicle routing problem," in *European Conference on Artificial Intelligence*, 2020.

[21] A. Hottung, B. Bhandari, and K. Tierney, "Learning a latent search space for routing problems using variational autoencoders," in *International Conference on Learning Representations*, 2021.

[22] W. Kool, H. van Hoof, J. Gromicho, and M. Welling, "Deep policy dynamic programming for vehicle routing problems," ArXiv, arXiv preprint arXiv:2102.11756, 2021.

[23] Y. Ma, Z. Cao, and Y. M. Chee, "Learning to search feasible and infeasible regions of routing problems with flexible neural k-opt," in *Advances in Neural Information Processing Systems*, vol. 36, 2023, pp. 49 555–49 578.

[24] Z. Sun and Y. Yang, "Difusco: Graph-based diffusion solvers for combinatorial optimization," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[25] J. Zhao, M. Mao, X. Zhao, and J. Zou, "A hybrid of deep reinforcement learning and local search for the vehicle routing problems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 11, pp. 7208–7218, 2020.

[26] R. García-Torres, A. A. Macias-Infante, S. E. Conant-Pablos, J. C. Ortiz-Bayliss, and H. Terashima-Marín, "Combining constructive and perturbative deep learning algorithms for the capacitated vehicle routing problem," *arXiv preprint arXiv:2211.13922*, 2022.

[27] M. W. Savelsbergh, "An efficient implementation of local search algorithms for constrained routing problems," *European Journal of Operational Research*, vol. 47, no. 1, pp. 75–85, 1990.

[28] J. Renaud, F. F. Boctor, and J. Ouenniche, "A heuristic for the pickup and delivery traveling salesman problem," *Computers & Operations Research*, vol. 27, no. 9, pp. 905–916, 2000.

[29] J. Renaud, F. F. Boctor, and G. Laporte, "Perturbation heuristics for the pickup and delivery traveling salesman problem," *Computers & Operations Research*, vol. 29, no. 9, pp. 1129–1141, 2002.

[30] M. Veenstra, K. J. Roodbergen, I. F. Vis, and L. C. Coelho, "The pickup and delivery traveling salesman problem with handling costs," *European Journal of Operational Research*, vol. 257, no. 1, pp. 118–132, 2017.

[31] F. Carrabs, J.-F. Cordeau, and G. Laporte, "Variable neighborhood search for the pickup and delivery traveling salesman problem with LIFO loading," *INFORMS Journal on Computing*, vol. 19, no. 4, pp. 618–632, 2007.

[32] F. Glover, "Tabu search—part i," *ORSA Journal on computing*, vol. 1, no. 3, pp. 190–206, 1989.

[33] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," ArXiv, arXiv preprint arXiv:1707.06347, 2017.

[34] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," in *International Conference on Machine Learning (Workshop)*, 2017.

[35] X. Wang, Y. Chen, and W. Zhu, "A survey on curriculum learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 9, pp. 4555–4576, 2021.

[36] A. Hottung, Y.-D. Kwon, and K. Tierney, "Efficient active search for combinatorial optimization problems," in *International Conference on Learning Representations*, 2022.

[37] S. Li, Z. Yan, and C. Wu, "Learning to delegate for large-scale vehicle routing," in *Advances in Neural Information Processing Systems*, vol. 34, 2021, pp. 26 198–26 211.
[38] L. Accorsi, A. Lodi, and D. Vigo, "Guidelines for the computational testing of machine learning approaches to vehicle routing problems," *Operations Research Letters*, vol. 50, no. 2, pp. 229–234, 2022.
[39] K. Zhou, Z. Liu, Y. Qiao, T. Xiang, and C. C. Loy, "Domain generalization: A survey," ArXiv, arXiv preprint arXiv:2103.02503, 2021.
[40] J. Li, Y. Ma, R. Gao, Z. Cao, A. Lim, W. Song, and J. Zhang, "Deep reinforcement learning for solving the heterogeneous capacitated vehicle routing problem," *IEEE Transactions on Cybernetics*, 2021.
[41] Y. Ma, X. Hao, J. Hao, J. Lu, X. Liu, T. Xialiang, M. Yuan, Z. Li, J. Tang, and Z. Meng, "A hierarchical reinforcement learning based optimization framework for large-scale dynamic pickup and delivery problems," in *Advances in Neural Information Processing Systems*, vol. 34, 2021, pp. 23 609–23 620.
[42] G. Erdoğan, J.-F. Cordeau, and G. Laporte, "The pickup and delivery traveling salesman problem with first-in-first-out loading," *Computers & Operations Research*, vol. 36, no. 6, pp. 1800–1808, 2009.

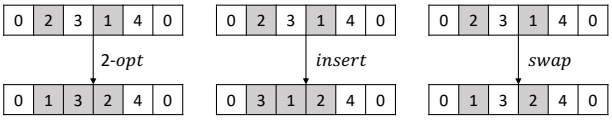
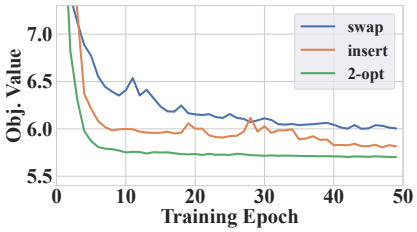


Fig. 11: Illustrative examples of three pairwise operators for routing problems when node pair ($i = 2$, $j = 1$) is specified for operating. From left to right: *2-opt*, *insert*, and *swap*.

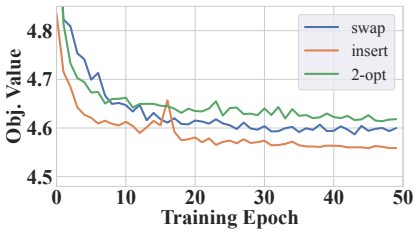
Appendix A
Comparison of DACT Operators for PDPs

As illustrated in Fig. 11, the prevalent neural improvement method DACT [5] features three types of operators in its decoder: *2-opt*, *swap*, and *insert*, with the *2-opt* operator demonstrating the best performance. Though the *2-opt* operator, considering reversing a segment of the solution, excels in classic VRPs (e.g., TSP and CVRP), it is not suitable for PDPs since the precedence constraint can be easily violated by segment inversion, thereby making DRL training difficult. The *swap* operator, focusing on exchanging the position of two nodes, typically represents the least effective option for both classic VRPs and PDPs. This is partly due to its decomposability into two insert steps in general (i.e., swapping node 1 with node 2 is equivalent to inserting node 1 after node 2, followed by inserting node 2 after the predecessor of node 1), resulting in less flexibility when compared to the *insert* operator for fine-grained search. The *insert* operator, which entails removing and reinserting a single node after another, shows the best performance among the three operators in small-scale PDPs. However, its effectiveness decreases in larger-scale or highly constrained PDPs (as shown in section 5.3).

In Fig. 12, the convergence curves of three operators in DACT are presented, demonstrating their performance during training on TSP-50 and PDTSP-21, respectively. The *2-opt* operator shows the best results for TSP, but its effectiveness diminishes in handling the precedence constraints in PDTSP. The *insert* operator consistently outperforms the *swap* operator for both TSP and PDTSP, making the *insert* operator the most preferred choice for employing DACT in solving PDPs.



(a) TSP



(b) PDTSP

Fig. 12: Convergence curves of three operators in DACT for TSP and PDTSP.

Appendix B
Mathematical formulations of PDTSP and PDTSP-LIFO

Following [42], we present the mathematical formulations of PDTSP and PDTSP-LIFO as follows. Let $V^{\text{PD}} = P \cup D$, $E^{\text{PD}} \subset E$ be the subset of edges with both endpoints in V^{PD}, $E^{\text{S}} := E \setminus E^{\text{PD}}$, and e_{ij} be the notation of edge $(x_i \to x_j)$ with Euclidean length c_{ij}, $K = \{1, \cdots, n\}$ be the set of transportation requests. The decision variables include $\mathbf{x}_{ij}$, $\mathbf{y}^1_{ijk}$, $\mathbf{y}^2_{ijk}$ and $\mathbf{y}^3_{ijk}$ defined as follows: the variable $\mathbf{x}_{ij}$ equals to one if and only if edge $e_{ij} \in E$ is in the current solution δ; the variable $\mathbf{y}^1_{ijk}$ equals to one if and only if edge $e_{ij} \in E$ is on the partial path from depot 0 to node k^+; the variable $\mathbf{y}^2_{ijk}$ equals to one if and only if edge $e_{ij} \in E$ is on the partial path from node k^+ to node k^-; the variable $\mathbf{y}^3_{ijk}$ equals to one if and only if edge $e_{ij} \in E$ is on the partial path from node k^- to depot 0. Accordingly, the PDTSP can be formulated as follows:

$$\text{Minimize} \sum_{x_i \in V} \sum_{x_j \in V} c_{ij} \mathbf{x}_{ij} \tag{21a}$$

subject to

$$\sum_{x_i \in V} \mathbf{x}_{iv} = 1 \ \ (\forall x_v \in V) \tag{21b}$$

$$\sum_{x_j \in V} \mathbf{x}_{vj} = 1 \ \ (\forall x_v \in V) \tag{21c}$$

$$\sum_{j \in V} \mathbf{y}^1_{ijk} - \sum_{j \in V} \mathbf{y}^1_{jik} = \begin{cases} 1 & \text{if } i = 0 \\ -1 & \text{if } i = k^+ \\ 0 & \text{otherwise} \end{cases} \quad (\forall x_i \in V,\ k \in K) \tag{21d}$$

$$\sum_{j \in V} \mathbf{y}^2_{ijk} - \sum_{j \in V} \mathbf{y}^2_{jik} = \begin{cases} 1 & \text{if } i = k^+ \\ -1 & \text{if } i = k^- \\ 0 & \text{otherwise} \end{cases} \quad (\forall x_i \in V^{\text{PD}},\ k \in K) \tag{21e}$$

$$\sum_{j \in V} \mathbf{y}^3_{ijk} - \sum_{j \in V} \mathbf{y}^3_{jik} = \begin{cases} 1 & \text{if } i = k^- \\ -1 & \text{if } i = 0 \\ 0 & \text{otherwise} \end{cases} \quad (\forall x_i \in V,\ k \in K) \tag{21f}$$

$$\mathbf{y}^1_{ijk} + \mathbf{y}^3_{ijk} = \mathbf{x}_{ij} \ \ (\forall e_{ij} \in E^{\text{S}},\ k \in K) \tag{21g}$$

$$\mathbf{y}^1_{ijk} + \mathbf{y}^2_{ijk} + \mathbf{y}^3_{ijk} = \mathbf{x}_{ij} \ \ (\forall e_{ij} \in E^{\text{PD}},\ k \in K) \tag{21h}$$

$$x_{ij} \in \{0, 1\} \ \ (\forall e_{ij} \in E) \tag{21i}$$

$$\mathbf{y}^1_{ijk}, \mathbf{y}^3_{ijk} \in \{0, 1\} \ \ (\forall e_{ij} \in E,\ k \in K) \tag{21j}$$

$$\mathbf{y}^2_{ijk} \in \{0, 1\} \ \ (\forall e_{ij} \in E^{\text{PD}},\ k \in K) \tag{21k}$$

The objective function (21a) minimizes the total route length. The constraint (21b) and (21c) are standard degree constraints. The constraint (21d) ensures that there is a partial path from the depot to each pickup node. The constraint (21e) ensures that there is a partial path from each pickup node to its pairing delivery node and also ensures that the pickup node is visited before the delivery node (the precedence constraint). The constraint (21f) ensures that there is a partial path from each delivery node back to the depot. The constraints (21d), (21e), and (21f) also eliminate sub-tours of the graph. The constraints (21g) and (21h) build relationships between individual decision variables for each partial path with the overall routing plan. The remaining constraints (21i), (21j), and (21k) describe the domain of each decision variable. The formulation of PDTSP-LIFO can be obtained by further including

TABLE 11: Descriptions of the four invariant transformations.

Transformations	Formulations	Configurations
flip-x-y	$(\hat{x}, \hat{y}) = (y, x)$	perform or skip
1-x	$(\hat{x}, \hat{y}) = (1 - x, y)$	perform or skip
1-y	$(\hat{x}, \hat{y}) = (x, 1 - y)$	perform or skip
rotate	$\begin{pmatrix}\hat{x}\\ \hat{y}\end{pmatrix} = \begin{pmatrix}x\cos\vartheta - y\sin\vartheta\\ x\sin\vartheta + y\cos\vartheta\end{pmatrix}$	$\vartheta \in \{0, \pi/2, \pi, 3\pi/2\}$

constraint (22) which states that in the partial path from the pickup node j^+ to the delivery node j^-, any additional goods collected (for instance, from the pickup node k^+) must be delivered before arriving at the delivery node j^-.

$$\sum_{e_{i,k^+} \in E^{\text{PD}}} \mathbf{y}^2_{i,k^+,j} = \sum_{e_{i,k^-} \in E^{\text{PD}}} \mathbf{y}^2_{i,k^-,j} \quad (\forall j, k \in K : j \neq k) \tag{22}$$

APPENDIX C
DETAILS OF ALGORITHMS AND HYPERPARAMETERS IN NCS

The details of *InvariantTransform*, *CurriculumLearning* and *ImitationLearning* used as functions in Algorithm 1 (main paper) are presented in Algorithm 3, 4 and 5, respectively. Algorithm 3 sequentially applies four preset invariant transformation operations with different orders and different configurations as listed in Table 11. In Algorithm 4, Γ_i, Γ_d, Θ_m, Θ_i, Υ_m, Υ_i are all hyperparameters which are used to regulate the curriculum learning phase. In Algorithm 5, Ψ_m, Ψ_w, Ψ_b are hyperparameters which are used to regulate the number of iterations. The setup of hyperparameters in curriculum learning and imitation learning of NCS are summarized in Table 12.

The detail of *PPO with shared-critic* used in Algorithm 1 is given in Algorithm 6, where we present the reinforcement learning loss function and the baseline loss function in Eq. (23) and Eq. (24) for the improvement policy and in Eq. (25) and Eq. (26) for the construction policy, respectively. We clip the estimated value $v_\phi(s_\tau)$ and $V'(\zeta)$ around the previous value estimates using $v^{clip}_\phi(s_\tau) = \text{clip}\left[v_\phi(s_\tau), v_{old}(s_\tau) - \varepsilon, v_{old}(s_\tau) + \varepsilon\right]$ and $V'_{clip}(\zeta) = \text{clip}\left[V'(\zeta), V'(\zeta_{old}) - \varepsilon, V'(\zeta_{old}) + \varepsilon\right]$. Here $V'(\zeta)$ is the critic value (i.e., it will be used for the m-AM), which is calculated using the shared-critic mechanism.

TABLE 12: Setup of hyperparameters in NCS

NCS stage	Hyper parameter	Problem size $\lvert V\rvert$		
		21	51	101
High-temperature CL ($e < e_{ts}$)	Γ_i	10	10	10
	Γ_d	0.94	0.93	0.92
Multi-sample CL ($e \geq e_{ts}$)	Θ_m	128	256	512
	Θ_i	1.1	1.2	1.3
Improvement CL ($\lvert V\rvert > 100$ and $e \geq e_{ps}$)	Υ_m	/	/	25
	Υ_i	/	/	2
Imitation learning	Ψ_m	10	25	25
	Ψ_w	1	1	1
	Ψ_b	0	0	-2

$$J_{RL}(\theta) = \frac{1}{n|\mathcal{D}_b|} \sum_{\mathcal{D}_b} \sum_{\tau=t}^{t+n} \min\left(\frac{\pi_\theta(a_\tau|s_\tau)}{\pi_{old}(a_\tau|s_\tau)}\hat{A}_\tau, \text{clip}\left[\frac{\pi_\theta(a_\tau|s_\tau)}{\pi_{old}(a_\tau|s_\tau)}, 1-\varepsilon, 1+\varepsilon\right]\hat{A}_\tau\right), \tag{23}$$

$$L_{BL}(\phi) = \frac{1}{n|\mathcal{D}_b|} \sum_{\mathcal{D}_b} \sum_{\tau=t}^{t+n} \max\left(\left|v_\phi(s_\tau) - \hat{R}_\tau\right|, \left|v^{clip}_\phi(s_\tau) - \hat{R}_\tau\right|\right)^2. \tag{24}$$

$$J'_{RL}(\theta') = \frac{1}{|\mathcal{D}_b|} \sum_{\mathcal{D}_b} \max\left(\frac{\pi_{\theta'}(\delta'|s'_0)}{\pi_{old'}(\delta'|s'_0)}A', \text{clip}\left[\frac{\pi_{\theta'}(\delta'|s'_0)}{\pi_{old'}(\delta'|s'_0)}, 1-\varepsilon, 1+\varepsilon\right]A'\right), \tag{25}$$

$$L'_{BL}(\zeta) = \frac{1}{|\mathcal{D}_b|} \sum_{\mathcal{D}_b} \max\left(|V'(\zeta) - f(\delta')|, \left|V'_{clip}(\zeta) - f(\delta')\right|\right)^2. \tag{26}$$

Algorithm 3 Invariant Transform

Input: Instance $\mathcal{I}$

$\mathcal{A}_i \leftarrow$ **RandomShuffle**([flip-x-y, 1-x, 1-y, rotate]);
for each augment method $j \in \mathcal{A}_i$ **do**
  $\varrho_j \leftarrow$ **RandomConfig**(j);
  $\mathcal{I}' \leftarrow$ perform augment j on $\mathcal{I}_i$ with config ϱ_j;
end for
return $\mathcal{I}'$;

Algorithm 4 Curriculum Learning

Input: Instance batch $\mathcal{D}_b$, improvement policy π_θ, construction policy $\pi_{\theta'}$, epoch e

$e_{ts} = \lceil \log_{\Gamma_d}(1/\Gamma_i) \rceil$, $e_{ps} = e_{ts} + \lceil \log_{\Theta_i} \Theta_m \rceil$;
if $e < e_{ts}$ **then**
  $\Gamma = \Gamma_i * (\Gamma_d)^e$;
  Sample one solution δ_0 on each instance of $\mathcal{D}_b$ via $\pi_{\theta'}$ with temperature softmax [34]:
  $\pi_{\theta'}(a_\tau = i|s_\tau) = \text{softmax}(\{u_i/\Gamma\}_{i=0}^{2n+1})$;
else
  $\Theta = \min(\Theta_m, (\Theta_i)^{e-e_{ts}})$;
  Sample Θ solutions and pick the shortest one as δ_0 on each instance of $\mathcal{D}_b$ via $\pi_{\theta'}$;
  if problem size > 100 and $e \geq e_{ps}$ **then**
    $\Upsilon = \min(\Upsilon_m, (e - e_{ps})/\Upsilon_i)$;
    Improve δ_0 for Υ steps via π_θ;
  end if
end if
return δ_0;

APPENDIX D
FULL RESULTS OF PAIRED T-TEST

In Table 3 and 4 of section 5.2, we use bold gaps to indicate that our methods significantly outperform the best neural method (i.e., Heter-POMO-A), and underlined gaps indicate that our methods significantly outperform LKH (10k). To prove our significance

Algorithm 5 Imitation Learning

Input: construction policy $\pi_{\theta'}$, imitation solution δ^*, constructed solution δ', Instance batch $\mathcal{D}_b$, epoch e

$\Psi = \max(0, \min(\Psi_m, \lfloor \Psi_w \cdot e + \Psi_b \rfloor))$;
$\xi = 1 \; if \; f(\delta^*) < f(\delta') \; else \; 0$;
for $i = 1, ..., \Psi$ **do**
  $\mathcal{D}'_b \leftarrow$ **InvariantTransform**$(\mathcal{D}_b)$;
  Get $\pi_{\theta'}(\delta^*|s'_0)$ on $\mathcal{D}'_b$;
  Compute imitation loss J'_{IL} using Eq. (20);
  $\theta' \leftarrow \theta' - \eta_{\theta'} \nabla J'_{IL}$;
end for
return updated policy $\pi_{\theta'}$

Algorithm 6 PPO with shared-critic

Input: improvement policy π_θ, construction policy $\pi_{\theta'}$, critic v_ϕ, shared-critic parameter ζ, PPO clipping threshold ε, learning rate η_θ, $\eta_{\theta'}$, η_ϕ, η_ζ, mini-batch κ, current step t, $(\delta', f(\delta'))$, $\{(s_\tau, a_\tau, r_\tau), f(\delta_\tau)\}_{\tau=t-n}^{t-1}$

$\pi_{old} \leftarrow \pi_\theta$, $v_{old} \leftarrow v_\phi$;
$\pi_{old'} \leftarrow \pi_{\theta'}$, $\zeta_{old} \leftarrow \zeta$;
for $k = 1$ to κ **do**
  $\hat{R}_t = v_\phi(s_t)$;
  for $\tau \in \{t-1, ..., t-n\}$ **do**
    $\hat{R}_\tau \leftarrow r_\tau + \gamma \hat{R}_{\tau+1}$;
    $\hat{A}_\tau \leftarrow \hat{R}_\tau - v_\phi(s_\tau)$;
  end for
  $V'(\zeta) \leftarrow \text{mean}(\{f(\delta_\tau) - \zeta v_\phi(s_\tau)\}_{\tau=t-n}^{t-1})$;
  $A' \leftarrow f(\delta') - V'(\zeta)$;
  Compute RL loss $J_{RL}(\theta), J'_{RL}(\theta')$ using Eq. (23, 25) and clipped critic loss $L_{BL}(\phi), L'_{BL}(\zeta)$ using Eq. (24, 26);
  $\theta \leftarrow \theta + \eta_\theta \nabla J_{RL}(\theta)$;
  $\theta' \leftarrow \theta' - \eta_{\theta'} \nabla J'_{RL}(\theta')$;
  $\phi \leftarrow \phi - \eta_\phi \nabla L_{BL}(\phi)$;
  $\zeta \leftarrow \zeta - \eta_\zeta \nabla L'_{BL}(\zeta)$;
end for

claim, we perform two-sided paired t-tests on 2,000 test instances and report p-values in Table 13. The alternative hypothesis is that the distributions underlying the samples are unequal. If the p-value is smaller than 5%, then we reject the null hypothesis of equal averages and accept the alternative hypothesis. Specifically, if the averaged objective value of our method is lower and the corresponding p-value is less than 5%, then we will add bold or underlined marks to the gap in Table 3 and 4.

APPENDIX E
FULL RESULTS OF GENERALIZATION

We present more details for our generalization evaluation on benchmark datasets. For Heter-POMO-A, N2S-A and NCS-A, the coordinates of instances are normalized to $[0, 1] \times [0, 1]$ as per training, and we adopted the "closest" model learned in Section 5 to infer the instances, e.g., models trained on PDP-21 are used to infer PDP-25 instances, and models trained on PDP-101 are used to infer instances with $|V| \geq 101$. We increase C to 10 in our N2S decoders during generalization since it boosts the performance. Full results of Table 10 are gathered in Table 14 (PDTSP) and Table 15 (PDTSP-LIFO). The gaps in Table 14 are computed w.r.t the optimal solutions and the gaps in Table 15 are computed w.r.t. heuristic baselines B1 [31] and B2 [4]. In all cases, our N2S-A and NCS-A significantly outperform Heter-POMO-A. In specific, our NCS-A achieves the best gaps of 0.00% ($|V| = 101$), and 1.93% ($|V| = 201$) on PDTSP while the gaps of Heter-POMO are 1.46% ($|V| = 101$) and 8.66% ($|V| = 201$); our NCS-A achieves best average gaps of 0.42% (w.r.t. B1) and 1.62% (w.r.t. B2) on PDTSP-LIFO while the gaps of Heter-POMO are 9.63% (w.r.t. B1) and 10.98% (w.r.t. B2). This further reveals the favorable generalization of our N2S and NCS over others.

APPENDIX F
DEALING WITH CAPACITY CONSTRAINT

Our N2S is generic to the capacity constraint, similar to DACT for handling capacity in CVRP [5]. Specifically, we can, 1) make copies of depots (i.e., dummy depots) so that N2S can search solutions with different numbers of vehicles automatically; 2) add capacity/demand features to NFEs; 3) mask out infeasible choices in the Reinsertion decoder; and 4) use diversity enhancement as usual since it only affects the node coordinates. Those procedures also hold for the improvement model in our NCS. Nevertheless, the capacity in PDP might not be so crucial as the vehicle may always alternatively load or unload the goods.

TABLE 13: P-values of two-sided paired t-test between our methods and LKH (10k), as well as our methods and Heter-POMO-A.

Method-1	Method-2	PDTSP			PDTSP-LIFO		
		21	51	101	21	51	101
LKH (10k)	N2S-A (1k)	0.9727	0.7769	0.0000	0.9916	0.0041	0.0000
	N2S-A (2k)	0.9756	0.9383	0.5317	0.9916	0.0019	0.0000
	N2S-A (3k)	0.9791	0.8224	0.0059	0.9916	0.0012	0.0000
	NCS-A (1k)	0.9773	0.7769	0.0000	0.9917	0.0041	0.0000
	NCS-A (2k)	0.9783	0.8207	0.4254	0.9917	0.0015	0.0000
	NCS-A (3k)	0.9785	0.7138	0.0067	0.9917	0.0010	0.0000
Heter-POMO-A (3k or 4k)	N2S-A (1k)	0.7729	0.0000	0.0000	0.7032	0.0004	0.0000
	N2S-A (2k)	0.7702	0.0000	0.0000	0.7031	0.0000	0.0008
	N2S-A (3k)	0.7668	0.0000	0.0000	0.7032	0.0000	0.0002
	NCS-A (1k)	0.7685	0.0000	0.0000	0.7031	0.0002	0.7518
	NCS-A (2k)	0.7076	0.0000	0.0000	0.7031	0.0000	0.0008
	NCS-A (3k)	0.7674	0.0000	0.0000	0.7031	0.0000	0.0000

TABLE 14: Generalization performance on benchmark instances from [29] for PDTSP using the trained model in Section 5.

Instances	$\|V\|$	Optimal	Heter-POMO-A (3k)		N2S-A (3k)		NCS-A (3k)		Heter-POMO-A (3k)		N2S-A (3k)		NCS-A (3k)	
			Avg. Cost ↓	Best Cost ↓	Avg. Cost ↓	Best Cost ↓	Avg. Cost ↓	Best Cost ↓	Avg. Gap(%) ↓	Best Gap(%) ↓	Avg. Gap(%) ↓	Best Gap(%) ↓	Avg. Gap(%) ↓	Best Gap(%) ↓
N101P1	101	799	824	820	**800**	**799**	**800**	**799**	3.13	2.63	**0.13**	**0.00**	**0.13**	**0.00**
N101P2	101	729	736	735	**730**	**729**	**730**	**729**	0.96	0.82	**0.14**	**0.00**	**0.14**	**0.00**
N101P3	101	748	751	751	**748**	**748**	**748**	**748**	0.40	0.40	**0.00**	**0.00**	**0.00**	**0.00**
N101P4	101	807	815	814	808	**807**	**807**	**807**	0.99	0.87	0.12	**0.00**	**0.00**	**0.00**
N101P5	101	783	794	791	**783**	**783**	**783**	**783**	1.40	1.02	**0.00**	**0.00**	**0.00**	**0.00**
N101P6	101	755	766	763	**755**	**755**	**755**	**755**	1.46	1.06	**0.00**	**0.00**	**0.00**	**0.00**
N101P7	101	767	787	787	768	**767**	**767**	**767**	2.61	2.61	0.13	**0.00**	**0.00**	**0.00**
N101P8	101	762	783	782	764	**762**	**762**	**762**	2.76	2.62	0.26	**0.00**	**0.00**	**0.00**
N101P9	101	766	**766**	**766**	**766**	**766**	**766**	**766**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
N101P10	101	754	774	773	**754**	**754**	**754**	**754**	2.65	2.52	**0.00**	**0.00**	**0.00**	**0.00**
Average		767.0*	779.6	778.2	767.6	**767.0***	**767.2**	**767.0***	1.64	1.46	0.08	**0.00**	**0.03**	**0.00**
N201P1	201	1039	1109	1095	1068	1059	**1054**	**1049**	6.74	5.39	2.79	1.92	**1.44**	**0.96**
N201P2	201	1086	1117	1114	**1102**	**1099**	1103	1100	2.85	2.58	**1.47**	**1.20**	1.57	1.29
N201P3	201	1070	1198	1177	1112	1107	**1110**	**1106**	11.96	10.00	3.93	3.46	**3.74**	**3.36**
N201P4	201	1050	1114	1108	1073	**1066**	**1069**	1067	6.10	5.52	2.19	**1.52**	**1.81**	1.62
N201P5	201	1052	1187	1169	1094	1078	**1066**	**1063**	12.83	11.12	3.99	2.47	**1.33**	**1.04**
N201P6	201	1059	1120	1111	**1077**	**1072**	1085	1080	5.76	4.91	**1.70**	**1.23**	2.46	1.98
N201P7	201	1037	1170	1164	1076	1065	**1055**	**1051**	12.83	12.25	3.76	2.70	**1.74**	**1.35**
N201P8	201	1079	1215	1196	1112	1108	**1100**	**1096**	12.60	10.84	3.06	2.69	**1.95**	**1.58**
N201P9	201	1050	1208	1198	**1076**	**1073**	1094	1093	15.05	14.10	**2.48**	**2.19**	4.19	4.10
N201P10	201	1085	1198	1192	1116	1112	**1109**	**1107**	10.41	9.86	2.86	2.49	**2.21**	**2.03**
Average		1060.7*	1163.6	1152.4	1090.6	1083.9	**1084.5**	**1081.2**	9.71	8.66	2.82	2.19	**2.24**	**1.93**

TABLE 15: Generalization performance on benchmark instances from [4] for PDTSP-LIFO using the trained model in Section 5.

Instances	$\|V\|$	B1(2007)	B2(2011)	Heter-POMO-A (3k)		N2S-A (3k)		NCS-A (3k)		Heter-POMO-A (3k)		N2S-A (3k)		NCS-A (3k)		Heter-POMO-A (3k)		N2S-A (3k)		NCS-A (3k)	
				Avg. Cost ↓	Best Cost ↓	Avg. Cost ↓	Best Cost ↓	Avg. Cost ↓	Best Cost ↓	Avg. Gap to B1(%) ↓	Best Gap to B1(%) ↓	Avg. Gap to B1(%) ↓	Best Gap to B1(%) ↓	Avg. Gap to B1(%) ↓	Best Gap to B1(%) ↓	Avg. Gap to B2(%) ↓	Best Gap to B2(%) ↓	Avg. Gap to B2(%) ↓	Best Gap to B2(%) ↓	Avg. Gap to B2(%) ↓	Best Gap to B2(%) ↓
brd14051	25	4682.2	4672.0	4705	4705	4680	**4672**	**4672**	**4672**	0.49	0.49	-0.05	**-0.22**	**-0.22**	**-0.22**	0.71	0.71	0.17	**0.00**	**0.00**	**0.00**
	51	7763.2	7740.0	8276	8201	7948	7828	**7845**	**7764**	6.61	5.64	2.38	0.83	**1.05**	**0.01**	6.93	5.96	2.69	1.14	**1.36**	**0.31**
	75	7309.1	7232.4	9059	8938	7775	**7554**	**7738**	7639	23.94	22.29	6.37	**3.35**	**5.87**	4.51	25.26	23.58	7.50	**4.45**	**6.99**	5.62
	101	10005.2	9735.0	13315	13000	10539	**10370**	**10458**	10404	33.08	29.93	5.34	**3.65**	**4.53**	3.99	36.77	33.54	8.26	**6.52**	**7.43**	6.87
pr1002	25	16221.0	16221.0	**16221**	**16221**	**16221**	**16221**	**16221**	**16221**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
	51	31187.7	30936.0	**30936**	**30936**	**30936**	**30936**	**30936**	**30936**	**-0.80**	**-0.80**	**-0.80**	**-0.80**	**-0.80**	**-0.80**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
	75	46911.0	46673.0	47404	47202	47284	46923	**47067**	**46700**	1.05	0.62	0.80	0.03	**0.33**	**-0.45**	1.57	1.13	1.31	0.54	**0.84**	**0.06**
	101	63611.1	61433.0	62569	62565	62787	62353	**62292**	**62096**	-1.64	-1.64	-1.30	-1.98	**-2.07**	**-2.38**	1.85	1.84	2.20	1.50	**1.40**	**1.08**
fnl4461	25	2168.0	2168.0	**2168**	**2168**	**2168**	**2168**	**2168**	**2168**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
	51	4020.0	4020.0	4038	4037	**4020**	**4020**	**4020**	**4020**	0.45	0.42	**0.00**	**0.00**	**0.00**	**0.00**	0.45	0.42	**0.00**	**0.00**	**0.00**	**0.00**
	75	5865.0	5739.0	6118	6038	5905	**5763**	**5775**	5768	4.31	2.95	0.68	**-1.74**	**-1.53**	-1.65	6.60	5.21	2.89	**0.42**	**0.63**	0.51
	101	8852.8	8562.0	9186	9065	8827	8702	**8680**	**8575**	3.76	2.40	-0.29	-1.70	**-1.95**	**-3.14**	7.29	5.87	3.10	1.64	**1.38**	**0.15**
d18512	25	4683.4	4672.0	4707	4705	4680	**4672**	**4672**	**4672**	0.50	0.46	-0.07	**-0.24**	**-0.24**	**-0.24**	0.75	0.71	0.17	**0.00**	**0.00**	**0.00**
	51	7565.6	7502.0	8215	8126	7696	**7519**	**7627**	7532	8.58	7.41	1.72	**-0.62**	**0.81**	-0.44	9.50	8.32	2.59	**0.23**	**1.67**	0.40
	75	8781.5	8629.0	10282	10215	**8884**	**8802**	8989	8946	17.09	16.32	**1.17**	**0.23**	2.36	1.87	19.16	18.38	**2.96**	**2.00**	4.17	3.67
	101	10332.4	10256.4	13499	13218	**10729**	**10555**	10928	10776	30.65	27.93	**3.84**	**2.15**	5.76	4.29	31.62	28.88	**4.61**	**2.91**	6.55	5.07
d15112	25	93981.0	93981.0	**93981**	**93981**	**93981**	**93981**	**93981**	**93981**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
	51	143575.2	142113.0	144120	143716	**142113**	**142113**	**142113**	**142113**	0.38	0.10	**-1.02**	**-1.02**	**-1.02**	**-1.02**	1.41	1.13	**0.00**	**0.00**	**0.00**	**0.00**
	75	201385.4	199047.8	206442	204944	**200004**	**199076**	200084	**199076**	2.51	1.77	**-0.69**	**-1.15**	-0.65	**-1.15**	3.71	2.96	**0.48**	**0.01**	0.52	**0.01**
	101	276876.8	266925.3	272784	273693	269160	267305	**269135**	**267001**	-1.48	-1.15	-2.79	-3.46	**-2.80**	**-3.57**	2.19	2.54	0.84	0.14	**0.83**	**0.03**
nrw1379	25	3194.8	3192.0	**3192**	**3192**	**3192**	**3192**	**3192**	**3192**	**-0.09**	**-0.09**	**-0.09**	**-0.09**	**-0.09**	**-0.09**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
	51	5095.0	5055.0	6369	6517	5086	5056	**5059**	**5055**	25.00	27.91	-0.18	-0.77	**-0.71**	**-0.79**	25.99	28.92	0.61	0.02	**0.08**	**0.00**
	75	6865.1	6831.0	9647	9272	7081	6960	**7005**	**6928**	40.52	35.06	3.14	1.38	**2.04**	**0.92**	41.22	35.73	3.66	1.89	**2.55**	**1.42**
	101	10197.5	9889.4	13898	13592	10330	**9996**	**10147**	10124	36.29	33.29	1.30	**-1.98**	**-0.50**	-0.72	40.53	37.44	4.46	**1.08**	**2.60**	2.37
Average		40880.4	40134.4	41713.8	41593.6	40501.0	40280.7	**40450.2**	**40265.0**	9.63	8.80	0.81	**-0.17**	**0.42**	-0.04	10.98	10.14	2.02	**1.02**	**1.62**	1.15