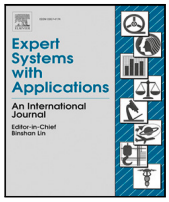
Expert Systems With Applications 255 (2024) 124514

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa



Solving pick-up and delivery problems via deep reinforcement learning based symmetric neural optimization

Jinqi Li [a], Yunyun Niu [a,*], Guodong Zhu [a], Jianhua Xiao [b]

[a] School of Information Engineering, China University of Geosciences, Beijing, 100083, China
[b] Research Center of Logistics, Nankai University, Tianjin, 300071, China

ARTICLE INFO

Dataset link: https://github.com/Good9T/PD-SNO

Keywords:
Deep reinforcement learning
Learning to optimize
Pick-up and delivery problem
Pick-up and delivery traveling salesman problem
Multi-commodity one-to-one pick-up and delivery traveling salesman problem
Attention mechanism

ABSTRACT

Recently, there has been a growing trend to use deep reinforcement learning (DRL) to solve NP-hard combinatorial optimization problems such as routing problem, where a policy learned by a deep neural network guides the sequential construction of solutions. Despite their success, most of the previous studies omit the symmetry between the partitions of instances and between solutions, which makes them less effective in handling the more practical scenarios with symmetric features, such as pick-up and delivery problems (PDPs). PDPs are more challenging and practical than traditional routing problems because the correlation between pick-up nodes and delivery nodes in the former is multiple one-to-one and not simple as the correlation in the latter. Besides, most DRL-based methods cannot take both computational cost and the quality of solutions into account. To resolve this issue, we propose the Symmetric Neural Optimization for Pick-up and Delivery Problems (PD-SNO) to solve the pick-up and delivery traveling salesman problem (PDTSP) and the multi-commodity one-to-one pick-up and delivery traveling salesman problem (m-PDTSP), respectively. The PD-SNO leverages the symmetries between different partitions of instances to effectively capture the relationship between symmetric node sets (i.e., pick-up nodes and delivery nodes); the PD-SNO also leverages the symmetry between different trajectories of solution to guide the construction of solutions and training of policy. We evaluate the performance of our PD-SNO, and the results show that it outperforms various representative baselines, including traditional methods (both exact and heuristic algorithms) and state-of-the-art DRL-based methods (both construction and improvement types) on synthetic datasets. Then, we further assess the generalization performance of our PD-SNO on benchmark datasets. Finally, we research the independent effectiveness of our designs through ablation studies.

1. Introduction

The pick-up and delivery problem (PDP) (Savelsbergh & Sol, 1995) is a classical and important combinatorial optimization problem (COP) (Cook, Lovász, Seymour, et al., 1995) in transportation and logistics. While the objective of the general PDP is to optimize the cost of finishing a number of pick-up and delivery requests, each request involves taking goods from each pick-up node and sending them to the paired delivery node (Yu, Aloina, Jodiawan, Gunawan, & Huang, 2023). PDP is more challenging to optimize than traditional routing problems such as the traveler salesman problem (TSP) because the relationship between the pick-up nodes and delivery nodes is multiple one-to-one and not as simple as the relationship in the TSP because the customers in TSP instances are homogeneous (Yıldız, Karaoğlan, & Altiparmak, 2023). Furthermore, the PDP can be expanded into many variants based on the general settings, with each variant characterized by unique aspects, including objectives, constraints, and other details (Parragh Sophie, Doerner Karl, & Hartl Richard, 2008). We study two representative PDP variants, i.e., the pick-up and delivery traveler salesman problem (PDTSP) (Mosheiov, 1994) and the multi-commodity one-to-one pick-up and delivery traveling salesman problem (m-PDTSP) (Hernández-Pérez & Salazar-González, 2009). The objective of the PDTSP is to optimize a Hamiltonian cycle within the general PDP constraints, and the nodes attending in the Hamiltonian cycle must include once of all pick-up nodes and delivery nodes and twice of the depot nodes (i.e., start and end) (Mosheiov, 1994); while the m-PDTSP is subjected to additional constraints, including the capacity of the vehicle and the demand of pick-up nodes and delivery nodes, where the demand of a pick-up node is the opposite number of the demand of its paired delivery node (Öztaş & Tuş, 2022). Both variants are widely used in scenarios of real-world logistics and transportation; one of the representative scenarios is that a deliveryman receives many orders

* Corresponding author.
E-mail addresses: jinqili@email.cugb.edu.cn (J. Li), yniu@cugb.edu.cn (Y. Niu), zgd@email.cugb.edu.cn (G. Zhu), jhxiao@nankai.edu.cn (J. Xiao).

https://doi.org/10.1016/j.eswa.2024.124514
Received 24 October 2023; Received in revised form 4 June 2024; Accepted 13 June 2024
Available online 27 June 2024

and needs to finish them with the least cost, where the contents of the orders are taking goods from the merchants (a.k.a., pick-up nodes) and delivering them to the designated customers (a.k.a., paired delivery nodes) (Hang Xu & Arunapuram, 2003).

Traditional methods are widely used for solving PDPs, which include exact and heuristic algorithms (Dubey & Tanksale, 2023). However, the requirements of high computational costs and handcrafted domain knowledge are still crucial challenges for traditional methods to yield higher optimal solutions due to the NP-hard essence of PDPs (Berbeglia Gerardo & Gilbert, 2007). By contrast, deep reinforcement learning (DRL)-based methods can obtain decent solutions without the aforementioned requirements, which attract the great concern of many researchers (Kalatzantonakis, Sifaleras, & Samaras, 2023). The DRL fuses the advantages of deep learning (DL) and reinforcement learning (RL), i.e., the DRL utilizes the deep neural network (from DL) to learn a policy for improving the ability of continuous decision-making (Bellman, 1957) (from RL), where the deep neural network is trained by a policy gradient (Kakade, 2001) (from RL) and the policy is based on complex attention mechanisms (Vaswani et al., 2017) (from DL). Since the construction or improvement of solutions in NP-hard COPs can also be cast as decision-making problems, it attracts many researchers to study DRL-based methods to obtain the solutions of COPs. Among them, the attention model (AM) (Kool, Van Hoof, & Welling, 2018) is one of the most widely recognized representatives of DRL-based methods for solving COPs, and it achieved promising performance on such problems as the traveler salesman problem (TSP) and the capacitated vehicle routing problem (CVRP), where the performance outperforms many traditional methods and its computational cost is exquisitely low. Specifically, the AM solved those COPs by fusing the Transformer model (Vaswani et al., 2017) (from DL) and the REINFORCE training algorithm (Williams, 1992) (from RL).

The DRL-based methods for solving COPs can be divided into two major types by the patterns of solution generation, i.e., construction and improvement, where the former sequentially constructs a complete solution, while the latter iteratively generates a new solution by improving the last solution (Wu, Fan, Cao, Gao, Hou, & Sartoretti, 2023). Typically, the DRL-based construction method can obtain better solutions with an exquisitely low computational cost, while the DRL-based improvement method can effectively leverage the high computational cost to generate a better solution. We focus on DRL-based construction methods in our work. In this series of works, the problem instances are first projected into an embedding, and then the embedding is processed into a probability distribution for selecting an action to sequentially construct the solution. Therefore, the computational cost of using DRL-based construction methods to solve COPs is primarily based on the times of actions taken, and the relationship between computational cost and the times of actions taken almost displays a linear correlation (Li et al., 2021). Besides, the DRL-based methods can be executed on GPU devices so that multiple solutions can be generated simultaneously, which can further reduce the computational cost. Thus, a major advantage of DRL-based construction methods over traditional methods is that the DRL-based methods can yield comparable (Kool et al., 2018) or superior (Kwon et al., 2020) results while running at an exquisitely low computational cost.

Although much success has been achieved, the DRL-based construction methods for PDPs are still insufficient in obtaining optimal solutions (Ma et al., 2022), as the prior works omit the characteristics of symmetry between different partitions of instances and between different trajectories that hindered them from yielding better solutions in PDPs. To resolve this issue, we propose **P**ick-up and **D**elivery with **S**ymmetry **N**eural **O**ptimization, i.e., PD-SNO. Specifically, we design a encoder–decoder architecture-based policy network and deploy it with multi-head symmetric attention (MHSA) mechanisms to effectively encode the relationship between symmetric partitions of instances (i.e., node sets of pick-up and delivery) and decode the construction of solutions (i.e., complete route), where the MHSA are respectively deployed in the encoder and the decoder simultaneously; we design a multi-query symmetric rollout and rotational symmetric augmentation to generate a number of different trajectories; we leverage the two aforementioned types of symmetric trajectories to form two types of loss functions; we design a combined loss function based on two different sub-loss functions to update the policy network; and we design an exchanged symmetric augmentation to further leverage the symmetry between partitions of instances during inference phases.

The performance of our PD-SNO is evaluated on a synthetic dataset containing 2000 test instances. We compare the performance with various representative baselines, including traditional methods (both exact and heuristic algorithms) and DRL-based methods (both construction and improvement types). A series of experimental results show that our PD-SNO outperforms those state-of-the-art baselines. Besides, we evaluate the generalization performance of our PD-SNO and the effectiveness of each design of our PD-SNO. Accordingly, the main contributions of this paper are shown as follows:

- A DRL policy network, namely PD-SNO, based on a multi-head symmetric attention mechanism is proposed to solve the PDTSP and the m-PDTSP, which is able to capture the relationship between symmetric partitions of instances and construct the solution by gathering the symmetric embeddings with the constantly updated environment.
- A multi-query symmetric rollout and a rotational symmetric augmentation is designed to generate the symmetric solution trajectories; a loss function based on those trajectories is developed to effectively training the policy network, and an exchanged symmetric augmentation is employed to yield a better solution during inference phase.
- A series of experiments on synthetic datasets and benchmark dataset is conducted to respectively testify the performance and generalization performance, and the experimental results show that the PD-SNO outperforms other state-of-the-art baselines. The ablation studies show that each design of the PD-SNO contributes to the performance.

The reminder of this paper are organized as follows: Section 2 reviews the traditional methods for solving PDPs and the DRL-based methods for solving COPs. Section 3 introduces the formulations for the PDTSP and m-PDTSP based on the mathematical model and Markov Decision Process (MDP). Section 4 introduces the details of our PD-SNO, including the architecture of the policy network, rollout, training, and inference. Section 5 shows and analyzes the experimental results. Section 6 concludes the work of this paper and envisions possible future works.

2. Related works

In this section, we review the traditional methods for solving PDPs and the DRL-based methods for solving COPs.

2.1. Traditional methods for solving PDPs

Traditional methods are widely utilized for solving PDPs. The PDP is first formulated as an integer program in Ruland and Rodin (1997), and a branch-and-cut algorithm based on the problem's constraints is presented to solve it. The multi-commodity PDTSP (m-PDTSP) is presented in Hernández-Pérez and Salazar-González (2009) and a decomposition technique is proposed to solve it. A heuristic approach that combines mathematical programming and meta-heuristic techniques proposed in Rodríguez-Martín and Salazar-González (2011) to solve the m-PDTSP. A real-world retail scenario formulated as the m-PDTSP in Li, Gong, Luo, and Lim (2019) and a branch-and-price-and-cut algorithm is proposed to solve it.

However, a series of studies show that these methods for solving PDPs heavily depend on handcrafted domain knowledge, which might

be unapproachable in some cases; besides, compared to the DRL-based methods, the higher computational cost of traditional methods for solving PDPs is always inevitable, which is inapt for contemporary requirements for swift computational results in real-world scenarios (Çağrı Koç, Laporte, & Tükenmez, 2020).

2.2. DRL-based methods for solving COPs

The attention model (AM) was proposed in Kool et al. (2018) to solve many COPs, including TSP, CVRP, and orienteering problems, and achieved near-optimal solutions; AM fuses the transformer structure in Vaswani et al. (2017) and the REINFORCE algorithm in Williams (1992) to utilize the self-attention mechanism and update the policy training. A DRL neural network based on the heterogeneous attention mechanism was proposed in Li et al. (2021) to solve the PDTSP and obtained results outperforming the AM. The Policy Optimization with Multiple Optima (POMO) was proposed in Kwon et al. (2020), which consists of a multiple optima rollout, a shared baseline, and a coordinate rotation augmentation. As a DRL-based construction method, POMO can produce better results for TSP and CVRP than the AM. However, it falls short of achieving comparable optimality for solving other COPs, such as PDTSP and m-PDTSP. A training scheme, Sym-NCO, was proposed in Kim, Park, and Park (2022) that leverages the invariant of problems and the symmetry of solutions; Sym-NCO outperforms other training schemes with a slight advantage in various COPs, including TSP, CVRP, prize collecting TSP, and orienteering problems. Except the DRL-construction methods, an efficient Neural Neighborhood Search (N2S) approach was proposed in Ma et al. (2022) to solve the PDTSP; the N2S is a DRL-based improvement method that features a synthesis attention and two decoders that autonomously perform removal and reinsertion of a pickup-delivery node pair separately; the N2S outperforms many baselines and became the first DRL-based method to outperform well-known LKH-3 solver (Helsgaun, 2017); despite the success in solution obtaining, the N2S needs a higher computational cost than DRL-based construction methods; the N2S is unable to select the neighborhood in m-PDTSP as effectively as in PDTSP, where the former has capacity constraint that a high proportion of the node pairs are masked in each step. While being able to solve various COPs, most of the aforementioned DRL-based methods failed to consider the universal symmetries in PDPs, including the symmetries between different partitions of instance and different solutions; besides, they have not utilized DRL-based methods to solve the PDPs with a capacity constraints, such as m-PDTSP, which is widely met in real-world scenarios.

3. Problem formulation

The formulations of PDTSP and m-PDTSP based on mathematical models and reinforcement learning models are introduced in this section, respectively.

3.1. The formulation of mathematical model

PDTSP. In PDTSP, one traveler will depart from the depot to satisfy all pick-up and delivery requests. Each request includes first visiting a pick-up node and then visiting its paired delivery node. Different pick-up and delivery requests can be mixed together. The traveler will return to the depot after satisfying all requests. Usually, a PDTSP instance contains a set $X = \{x_i\}_{i=0}^{2n}$ with $2n+1$ overall nodes, where their indices can be denoted as $X' = [0, 2n]$. The x_0 is the depot node, and the subset $x_p = \{x_i\}_{i=1}^{n}$ comprises n pick-up nodes, where their indices can be denoted as $X_p{}' = [1, n]$. The other subset $X_d = \{x_i\}_{i=n+1}^{2n}$ comprises n delivery nodes, where their indices can be denoted as $X_d{}' = [n, 2n]$. The set of customers X_c includes both the pick-up nodes and delivery nodes, where their indices can be denoted as $X_c{}' = X_p{}' + X_d{}' = [1, 2n]$. Each request consists of a pick-up node x_i and a paired delivery node x_{i+n}. Each node x_i is featured by its two-dimensional location coordinates G_i. Let $Z(x_i, x_j)$ denote the Euclidean distance from node x_i to node x_j, L_i denote the total route length that the traveler has traveled to node x_i. Then the binary decision variable $v_{ij} = 1$ denotes that the node x_j is visited directly after x_i, and $v_{ij} = 0$ otherwise. The objective of PDTSP is to find the shortest Hamiltonian cycle that includes all request nodes and meets the constraints that a pick-up node must be visited before its delivery node, which is defined as

$$\min \sum_{i \in X'} \sum_{j \in X'} Z(x_i, x_j) v_{ij}. \tag{1}$$

The PDTSP should meet the constraints as follows,

$$\sum_{i \in X'} v_{ij} = 1, \quad j \in X' \tag{2}$$

$$\sum_{j \in X'} v_{ij} = 1, \quad i \in X' \tag{3}$$

$$\sum_{i \in X'} v_{ij} - \sum_{k \in X'} v_{jk} = 0, \quad j \in X' \tag{4}$$

$$L_{i+n} \geq L_i + Z(x_i, x_{i+n}), \quad i \in X_p{}' \tag{5}$$

$$v_{ij} = \{0, 1\}, \quad i, j \in X' \tag{6}$$

$$L_i \geq 0, \quad i \in X' \tag{7}$$

Specifically, constraints (2) and (3) ensure that each node can be visited exactly once. Constraint (4) ensures that the route is continuous. Constraint (5) ensures that the pick-up node must be visited before its paired delivery node. Constraint (6) defines the binary decision variable, and constraint (7) defines the non-negativity of the total route length.

m-PDTSP. In m-PDTSP, the vehicle (traveler) is additionally subjected to the constraint of capacity; it departs from the depot without any loading; consumes the remaining capacity according to the demand of the goods while visiting a pick-up node; and releases the occupied capacity according to the demand of the goods while visiting a delivery node, where the demand of a pick-up node is the same in absolute value as the demand of the paired delivery node. Each node x_i is featured by (G_i, D_i), where G_i denotes two-dimensional location coordinates and D_i denotes demand, where the demand of pick-up nodes is positive numbers, while the demand of delivery nodes is positive numbers. Besides, the absolute value of demand of a pick-up node is the same as the absolute value of demand of the paired delivery node. Let C_{ij} denote the remaining capacity of the vehicle before traveling from node x_i to node x_j, while the initial capacity before departure from the depot is set as $C_{0j} = 0$ ($j \in [1, n], j \in \mathbb{N}$), and the maximum capacity of the vehicle is set as C_{max}. In addition to aforementioned constraints in PDTSP, the constraints in m-PDTSP are introduced as follows,

$$\sum_{i \in X'} C_{ij} - \sum_{k \in X'} C_{jk} = D_j, \quad j \in X_c{}' \tag{8}$$

Specifically, constraint (8) guarantees that the difference of remaining capacity before and after the vehicle serves a node equals its demand, whether it is a pick-up node or a delivery node.

3.2. RL formulation

The process of constructing the Hamiltonian cycle of PDTSP or m-PDTSP could be cast as a continuous decision-making problem. Therefore, the representation for PDTSP or m-PDTSP can be defined as a Markov Decision Process (MDP) (Bellman, 1957). Specifically, the MDP is defined by a 4-tuple $M = \{S, A, T, R\}$, i.e., S indicates the state

space, A indicates the action space, T indicates the transition rule, and R indicates the reward.

State. The state $S^t = (M^t, X^t)$ ($S^t \in S$) consists of two components. The first component of the PDTSP is $M^t = (L^t, O^t)$ denotes the traveler state, where L^t denotes the total route length that the traveler has traveled till step t, and $O^t = \{x_i^0, x_j^1, ..., x_k^t\}$ represents the traveled route that consist of visited node $\{x_i, x_j, ..., x_k\}$ and the corresponding step $\{0, 1, ..., t\}$. The second component of PDTSP is $X^t = \{X_i^t\}_{i=0}^{2n} = \{(G_i, l_i^t)\}_{i=0}^{2n}$ denotes the node state, where G_i represents the node position of x_i as two-dimensional coordinates, and the binary variable $l_i^t = 1$ denotes the node is locked according to the constraint (5) and the $l_i^t = 0$ otherwise, i.e., the $l_i^t = 1(i \in [n+1, 2n], i \in \mathbb{N})$ indicates that all delivery nodes are locked at the initial state because their paired pick-up nodes have not been visited. In m-PDTSP, the first component is the vehicle state $M^t = (C^t, L^t, O^t)$ similar to the traveler state in PDTSP, where the additional variable C^t denotes the remaining capacity of the vehicle on step t. The initial vehicle state is set as $M^0 = (0, 0, \{x_i^0\})$ when the vehicle departs from the depot. The second component of m-PDTSP is node state $X^t = \{x_i^t\}_{i=0}^{2n} = \{(G_i, l_i^t, D_i^t)\}_{i=0}^{2n}$ similar to the node state in PDTSP, where the additional variable D_i^t denotes the demand of node x_i at step t.

Action. The action in PDTSP or m-PDTSP is defined as a step in the construction of the Hamiltonian cycle, which is guided by the RL's policy. Concretely, the action A^t ($A^t \in A$) is represented as x_i^t, i.e., the traveler visits the node x_i at step t.

Transition. A state S^t will be changed to the next state S^{t+1} by the transition rule T based on the action $A^t = x_i^t$ and its successive action $A^{t+1} = x_j^{t+1}$. In PDTSP, the traveler state M^t is updated to $M^{t+1} = \{L^{t+1}, O^{t+1}\}$ as follows,

$$L^{t+1} = L^t + Z(x_i, x_j), \tag{9}$$

$$O^{t+1} = [O^t : x_j^{t+1}], \tag{10}$$

where $[O^t : x_j^{t+1}]$ denotes the concatenation of the O^t and the x_j^{t+1}, i.e., the traveled route after the A^{t+1}. The node state X^t is updated to $X^{t+1} = \{G_i, l_i^{t+1}\}$ as follows,

$$l_{j+n}^{t+1} = 0, \quad \text{if } 0 < j \le n \tag{11}$$

While in m-PDTSP, the vehicle state M^t is updated to $M^{t+1} = \{C^{t+1}, L^{t+1}, O^{t+1}\}$ is similar to the traveler state in PDTSP, and the additional variable C^t is updated to C^{t+1} as follows,

$$C^{t+1} = C^t + D_j \tag{12}$$

The node state X^t is updated to $X^{t+1} = \{G_i, D_i, l_i^{t+1}\}$ is similar to the node state in PDTSP.

Reward. The reward R represented by the negative of the total length L of the completed Hamiltonian cycle as follows,

$$R = -L^f, \tag{13}$$

where f denotes the index of the finished state.

Policy. The stochastic policy π_θ governing the action at each step from the initial state S^0 to the finished state S^f, which is the goal of RL and learned by the deep neural network. The policy can be represented as a joint probability distribution of all actions in the construction of Hamiltonian cycle as follows,

$$P(S^f | S^0) = \prod_{t=0}^{f-1} \pi_\theta(A^t | S^t). \tag{14}$$

4. Methodology

The PD-SNO (pick-up and Delivery with Symmetric Neural Optimization) is introduced in this section, including its network's architecture, rollout, training, and inference.

4.1. Architecture

Overview. The architecture of PD-SNO is shown in Fig. 1 is composed of an encoder and a decoder. The encoder first uses linear projections to process the instance into an embedding and send the embedding to multiple identical sub-layers, which can learn the relationship between symmetric node sets. Obtaining symmetry between different partitions of instances (i.e., pick-up nodes and delivery nodes) is important. This symmetry can be represented by a number of attention scores, which are generated by the query and key belonging to different node sets in encoder sub-layers. Conceptually, the symmetry between different node sets is similar to company operation, which can only run well if every employee works in their specific post appropriately. In prior works, the encoder sub-layers processed the instance as an overall embedding and performed a multi-head self-attention mechanism on it. Admittedly, these works can adapt well to TSP or CVRP because the customer nodes in these problems have only one role. In contrast, the encoder sub-layers of PD-SNO split the embedding by the heterogeneous role of the node and perform the multi-head-symmetric attention (MHSA) mechanism on different embeddings. Afterwards, the sub-layers use normalization and feed-forward operations to process the generated attention. The embeddings are passed to the decoder after being encoded sequentially by all encoder sub-layers as encoded pick-up nodes, encoded delivery nodes, and encoded overall nodes. In the decoder, PD-SNO leverages the symmetry between the encoded pick-up nodes and the encoded delivery nodes and performs the MHSA on them, then interacts the processed embeddings with the environment to select the next node to visit. Particularly, the decoder first uses the encoded overall nodes and the step state of the environment to generate the query, then uses the generated query to calculate the attention of pick-up nodes, delivery nodes, and overall nodes and concatenates them to generate the output of MHSA that is further transformed to the probability for node selection after a series of processes such as scaling, clipping, masking, and softmax. Besides, the key and value are generated by linearly projecting the encoded pick-up nodes, delivery nodes, and overall nodes in the initial state and reused at each step to reduce the computational cost. We will display the details of the architecture in the following:

Encoder with a Multi-Head Symmetric Attention Mechanism. The encoder is the first module of the PD-SNO. Considering that pick-up nodes and delivery nodes play heterogeneous roles in PDTSP, PD-SNO uses linear projections to transform the information of overall nodes I^n, pick-up nodes I^p, and delivery nodes I^d into three embeddings for a more comprehensive capture of the relationship between different types of nodes, i.e. the three embeddings include the embedding of overall nodes E^n, pick-up nodes E^p, and delivery nodes E^d and generated as follows,

$$E^e = I^e W^{E^e}, \tag{15}$$

with $\forall e \in \{n, p, d\}$, and W^{E^e} are trainable parameters. Afterwards, three embeddings are processed through a series of U (U=6) sub-layers, which start with the Multi-head-symmetric attention (MHSA) mechanism. In PDTSP, the relationships from node to node are different from those in TSP due to the heterogeneous roles of the pick-up nodes and the delivery nodes. Compared to the prior works, the encoder sub-layers with the MHSA process embeddings based on the characteristics of PDTSP include the heterogeneity between the pick-up nodes and the delivery nodes as well as the symmetry between the set of pick-up nodes and the set of delivery nodes. The heads number of the MHSA mechanism is set to Y (Y=16). The dimension of the embeddings dim_{embed} is set as 256, and the dimensions of query, key, and value are set as $dim_{qkv} = \frac{dim_{embed}}{Y}$. Specifically, MHSA first linearly projects the embedding of overall nodes E^n, pick-up nodes E^p, and delivery nodes E^d to generate the query of overall nodes Q_{uy}^n, query of pick-up nodes

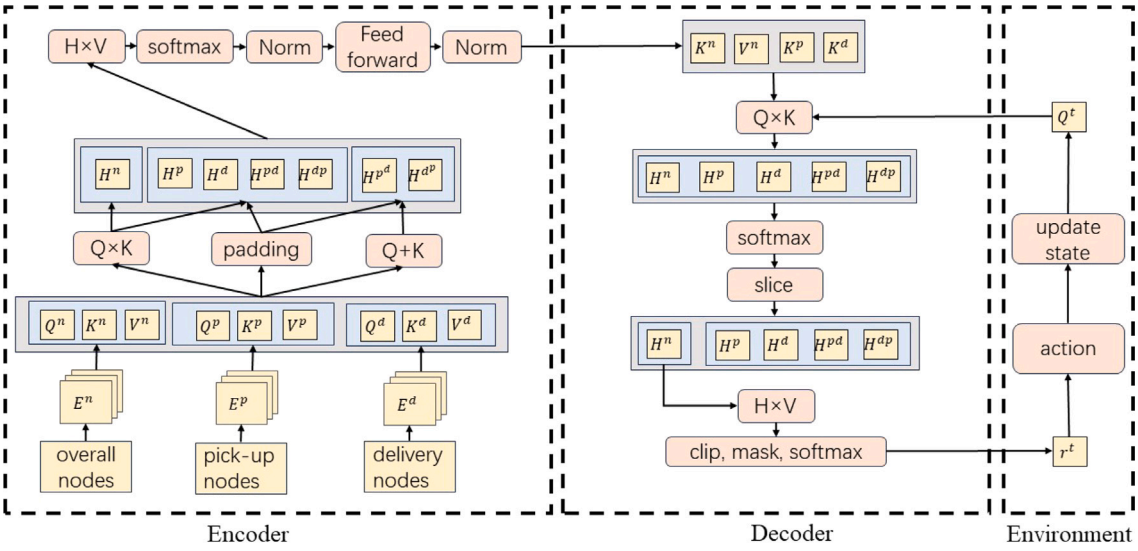


Fig. 1. The architecture of PD-SNO.

Q^p_{uy}, query of delivery nodes Q^d_{uy}, key of overall nodes K^n_{uy}, key of pick-up nodes K^p_{uy}, key of delivery nodes K^d_{uy}, value of overall nodes V^n_{uy}, value of pick-up nodes V^p_{uy}, and value of delivery nodes V^d_{uy} as follows,

$$Q^e_{uy} = E^e_u W^{Q^e_u}, \tag{16}$$

$$K^e_{uy} = E^e_u W^{K^e_u}, \tag{17}$$

$$V^e_{uy} = E^e_u W^{V^e_u}, \tag{18}$$

with $\forall u \in \{1, ..., U\}$, $\forall y \in \{1, ..., Y\}$, $\forall e \in \{n, p, d\}$, where E^e_u denotes the input of the u-th encoder sub-layer for e type embedding; $W^{Q^e_u}$, $W^{K^e_u}$, $W^{V^e_u}$ are trainable parameters in each u-th encoder sub-layer. Except for the basic value, MHSA additionally generates values $V^{p'}_{uy}$ and $V^{d'}_{uy}$ by padding the dimension of V^d_{uy} and V^p_{uy} with 0 tensors to the same shape as V^n_{uy} for calculating the attention from pick-up nodes to delivery nodes and from delivery nodes to pick-up nodes. Afterwards, MHSA calculate the scale dot-product attention (SDPA) of overall nodes H^n_{uy}, pick-up nodes H^p_{uy}, and delivery nodes H^d_{uy} as follows,

$$H^e_{uy} = Q^e_{uy}(K^e_{uy})^\top \tag{19}$$

with $\forall u \in \{1, ..., U\}$, $\forall y \in \{1, ..., Y\}$, $\forall e \in \{n, p, d\}$. Similarly, meditating the symmetry between the set of pick-up nodes and the set of delivery nodes, MHSA calculate the attention from the pick-up nodes to all delivery nodes H^{pd}_{uy} and delivery nodes to all pick-up nodes H^{dp}_{uy} as follows,

$$H^{pd}_{uy} = Q^p_{uy}(K^d_{uy})^\top \tag{20}$$

$$H^{dp}_{uy} = Q^d_{uy}(K^p_{uy})^\top \tag{21}$$

Additionally, considering the symmetry between pick-up nodes and delivery nodes, MHSA calculate the attention from each pick-up node to its paired delivery node $H^{p^d}_{uy}$ and from each delivery node to its paired pick-up node $H^{d^p}_{uy}$ as follows,

$$H^{p^d}_{uy} = Q^p_{uy} + K^d_{uy} \tag{22}$$

$$H^{d^p}_{uy} = Q^d_{uy} + K^p_{uy} \tag{23}$$

Each aforementioned attention is padded with 0 tensors to the same shape as the SDPA of overall nodes H^n_{uy}, concatenated together, and performed a softmax to generate the concatenate attention H^+_{uy} as follows,

$$H^+_{uy} = \text{softmax}(H^n_{uy} + H^{p^d}_{uy} + H^p_{uy} + H^{pd}_{uy} + H^{d^p}_{uy} + H^d_{uy} + H^{dp}_{uy}) \tag{24}$$

Then, the concatenated attention will be sliced into its original shape as those attentions before concatenated. Those sliced attentions are processed as similar as SDPA processed in self-attention mechanism, i.e. those attentions matrix-multiply or dot-product with the corresponding value, and the results concatenated again into the MHSA output H^s_{uy} as follows,

$$H^s_{uy} = \sum_e H^e_{uy} V^e_{uy} + H^{p^d}_{uy} * V^{d'}_{uy} + H^{d^p}_{uy} * V^{p'}_{uy} + H^{pd}_{uy} V^d_{uy} + H^{dp}_{uy} V^p_{uy} \tag{25}$$

with $\forall u \in \{1, ..., U\}$, $\forall y \in \{1, ..., Y\}$, $\forall e \in \{n, p, d\}$, where * is the dot-product operator. The MHSA output H^s_{uy} for each head will be processed by a multi-head combine layer, two normalization layers and a feed-forward layer as follows,

$$H^{s,+}_u = H^s_{uy} W^{H^{s,+}_u} \tag{26}$$

$$F^1_u = \text{norm}(H^{s,+}_u + E^e_u), \tag{27}$$

$$F^2_u = \text{relu}(F^1_u W^{F^1_u}) W^{F^2_u}, \tag{28}$$

$$F^3_u = \text{norm}(F^1_u + F^2_u), \tag{29}$$

with $\forall u \in \{1, ..., U\}$, $y \in \{1, ..., Y\}$, where $H^{s,+}_u$ denotes the output of the multi-head combine layer, F^1_u denotes the output of the first normalization, F^2_u denotes the feed-forward output, F^3_u denotes the output of second normalization, $W^{H^{s,+}_u}$, $W^{F^1_u}$ and $W^{F^2_u}$ are trainable parameters. The output of the last encoder sub-layer is the encoded embedding of overall nodes E^n_z, and it will be split into the encoded embedding of pick-up nodes E^p_z, and delivery nodes E^d_z, where z denotes that these variable are used in the decoder.

Decoder with a Multi-Head-Symmetric Attention Mechanism. In the PDTSP, the set of pick-up nodes and the set of delivery nodes are symmetric; therefore, the relationship between pick-up nodes and delivery nodes cannot be simply deemed the same as the node relationship in the TSP, which may limit the deep neural network's ability to yield high-quality solutions. To address this issue, we design a decoder with a MHSA to simultaneously compute and concatenate the attentions of different types of nodes, including overall nodes, pick-up nodes, and delivery nodes. As Fig. 1 displays, the decoder has a similar yet simpler architecture as the encoder due to the fact that the decoder will repeat the run (i.e., node selection) until the complete construction of the Hamiltonian cycle and interact with the environment frequently. The decoder will be initialized after the encoder's work is done and receive

the embedding of overall nodes E_z^n from the encoder. The decoder will generate addition embedding by slicing the encoded embedding E_z^n of overall nodes into the encoded embedding of pick-up nodes E_z^p and delivery nodes E_z^d, where z indicates it is present in the decoder to distinguish it from the same notation in the encoder. With the similar architecture as the encoder, the number of heads (Y) as well as the shape of query, key, and value (dim_{qkv}) in the decoder are the same as those in the encoder. The decoder then yields the key of overall nodes K_{zy}^n, pick-up nodes K_{zy}^p, delivery nodes K_{zy}^d, and value of overall nodes V_{zy}^n and finishes the initialization as follows,

$$K_{zy}^e = E_z^e W^{K_{zy}^e}, \tag{30}$$

$$V_{zy}^n = E_z^n W^{V_{zy}^n}, \tag{31}$$

with $\forall y \in \{1, ..., Y\}$, $\forall e \in \{n, p, d\}$, where $W^{K_{zy}^e}$ and $W^{V_{zy}^n}$ are trainable parameters. The running of the decoder will last until it reaches the finish state, i.e., the complete construction of the PDTSP Hamiltonian cycle. Firstly, the decoder generates the query by using the node index of last selection in the step state S^t from environment to select the node in the encoded embedding of overall nodes E_z^n from encoder as follows,

$$Q_{zy}^t = \text{select}(E_z^n, S^t) W^{Q_{zy}^t}, \tag{32}$$

with $\forall y \in \{1, \ldots, Y\}$, where t is the index of step state, and $W^{Q_{zy}^t}$ is a trainable parameter. Considering the symmetry between the set of pick-up nodes and delivery nodes, the decoder yields three SPDAs for node to overall nodes $H_{zy}^{nn,t}$, node to pick-up nodes $H_{zy}^{np,t}$, and node to delivery nodes $H_{zy}^{nd,t}$, respectively, as follows,

$$H_{zy}^{ne,t} = Q_{zy}^t (K_{zy}^e)^\top \tag{33}$$

with $\forall y \in \{1, ..., Y\}$, $\forall e \in \{n, p, d\}$. Then the SDPAs are masked by the constraints. Specifically, the mask will be sliced to adapt the shape of each SDPA as the mask of overall nodes $\text{mask}^{nn,t}$, pick-up nodes $\text{mask}^{np,t}$, and delivery nodes $\text{mask}^{nd,t}$.

$$H_{zy}^{ne,t} = H_{zy}^{ne,t} + \text{mask}^{ne,t} \tag{34}$$

with $\forall y \in \{1, \ldots, Y\}$, $\forall e \in \{n, p, d\}$. Regarding the mask rules in PDTSP, the delivery nodes are masked at initial state because their paired pick-up nodes have not been visited; the visited nodes are masked; and the delivery nodes with a visited paired pick-up node are unmasked. Generally, the PD-SNO masks and unmasks the node according to those rules at each step to ensure the feasibility of the constraints. Similar to the MHSA in the encoder, the SDPAs will be concatenated together, performed a softmax operation, sliced again yet without computing the heterogeneous relationship in decoder due to redundant computational cost, replaced with directly yielding the $H_{zy}^{nn,t}$ and matrix-multiply it with the value of overall nodes V_{zy}^n as the multi-head symmetric attention (MHSA) output $H_{zy}^{s,t}$ as follows,

$$H_{zy}^{s,t} = \frac{\text{slice}(\text{softmax}(\sum_e^{n,p,d} H_{zy}^{ne,t}), 0 : H_{zy}^{nn,t})}{\sqrt{dim_{embed}}} V_{zy}^n \tag{35}$$

with $\forall y \in \{1, \ldots, Y\}$, where $\text{slice}(A, 0 : B)$ is an operator indicates that slice the A and obtain the partition from 0 to B. Similar to the MHSA in the encoder, each MHSA output of the head will be combined together, matrix-multiplied with the embedding of overall nodes E_z^n, clipped with a tanh activation function, masked again, and softmaxed again to form the probability list r^t as follows,

$$H_z^{s,t+} = H_{zy}^{s,t} W^{H_z^{s,t+}} E_z^n \tag{36}$$

$$r^t = \text{softmax}(\text{CLIP} \times \tanh(H_z^{s,t+}) + \text{mask}^{nn,t}) \tag{37}$$

with $\forall y \in \{1, ..., Y\}$, where $W^{H_z^{s,t+}}$ is a trainable parameter, CLIP is a constant hyper-parameter. At the end of the step, the probability list r^t is used for node selection.

4.2. Symmetric rollout

Obtaining symmetry between different trajectories is important. This symmetry can be represented by the standard deviation of the total route length for each trajectory, which is different from the symmetry between partitions of instances. In our PD-SNO, this symmetry has been cast as a loss function. The lower the loss value, the more symmetric the trajectories are. After appropriate training, the loss value will be reduced, which means that the robustness of the policy network has been improved. Conceptually, the training based on the symmetry trajectories is similar to a writing class, where a teacher assigns the same writing work to all students. The students always come out with unique contents and various levels of work because they have different life experiences and writing bases with each other. The goal of the class is to improve the overall writing skills of each student, and the teacher takes the best work of all students as an example to teach others. As a result, each student can come out with relatively good work. When solving practical problems (the inference phase), the teacher takes the best work of all students to represent the writing skill of the class. To construct the Hamiltonian cycle, our PD-SNO utilizes a multi-query symmetric rollout and a rotational symmetric augmentation. Considering the rollout strategies, the greedy rollout strategy and the sampling rollout strategy are deployed in PD-SNO separately. The multi-query symmetric rollout, the rotational symmetric augmentation, the greedy rollout strategy, and the sampling rollout strategy are introduced as follows.

Multi-Query Symmetric Rollout. We use multi-query symmetric rollout in our PD-SNO to leverage the symmetry between different trajectories of solution. Different from prior works that randomly choose the first pick-up node to visit, PD-SNO simultaneously constructs n (i.e., n is the number of pick-up nodes) solutions, and we impose those solutions to visit different pick-up nodes as their second node to visit after departure from the depot. In each step, the query of the decoder is generated n times simultaneously, which is mentioned in Eq. (32). Each trajectory is defined as follows,

$$\tau_j = \{A_j^0, A_j^1, \ldots, A_j^{2n+1}\}, \quad \forall j \in \{1, \ldots, n\} \tag{38}$$

where $2n + 1$ is the number of steps for the trajectory τ_j to reach the terminal state on PDTSP. Each solution is constructed independently without corresponding with others. PD-SNO obtains the best solution in all trajectories as the final solution of the problem instance.

Rotational Symmetric Augmentation. Instance augmentation is usually employed to improve the solutions during the inference phase. Different from prior works, we use instance augmentation to generate a number of symmetric problem instances during both training and inference. Specifically, we employ coordinate rotational augmentation to generate the following 7 rotations based on the original $(x,\ y)$, including $(x,\ 1-y)$, $(1-x,\ y)$, $(1-x,\ 1-y)$, $(y,\ x)$, $(y,\ 1-x)$, $(1-y,\ x)$, $(1-y,\ 1-x)$. Therefore, the number of augmentations is $g = 8$. Each augmented instance is symmetric with the original instance and other augmented instances. Those augmented instances are definitely different from the original ones in the view of the policy network, but their solution can be essentially regarded as the same one because the Euclidean distances between each node are the same as the original instance.

Greedy Rollout Strategy. The greedy rollout strategy is one of the most effective rollout strategies for DRL-based construction methods because it always selects the node with the highest probability to visit; therefore, even if the inference is repeated multiple times, the inference results will always be the same for one instance. As a result, the greedy rollout strategy obtains an acceptable solution at an exquisitely low computational cost, as the inference only needs to perform a single forward pass of the neural network. For a more concise representation,

we use 'g.' to denote the DRL-based method using the greedy strategy for rollout.

Sampling Rollout Strategy. The sampling strategy is another effective rollout strategy for DRL-based construction methods, which always samples a node according to the probabilities; therefore, the selection may be varied while the inference is repeated multiple times, and each unmasked node has a chance to be selected to be visited. As a result, the sampling rollout strategy pursues a higher optimal solution by increasing the computational cost. For a more concise representation, we use 's.' to denote the DRL-based method using the sampling strategy for rollout, and we use 's.200' to indicate the case of sampling 200 solutions per (augmented) instance.

4.3. Symmetric training

During training of the PD-SNO, the aforementioned multi-query symmetric rollout and rotational symmetric augmentation are important to exploit the symmetry. Therefore, the query-symmetric loss and the augmentation-symmetric loss are introduced based on the multi-query symmetric rollout and the rotational symmetric augmentation, separately. The final loss is combined by both loss functions to efficiently guide the training. The query-symmetric loss, the augmentation-symmetric loss, the final loss, and the overall training algorithm are introduced in this section.

Multi-Query Symmetric Loss. The multi-query symmetric loss is used to evaluate the performance of training with multi-query symmetric rollout. Each query corresponds to a trajectory. After all trajectories finish the rollout, PD-SNO computes the mean reward of all multi-query trajectories and uses all the rewards of multi-query trajectories to subtract the mean reward to gain an advantage. The positive advantage can decrease the multi-query symmetric loss, while the negative advantage increases it. The mean reward of all multi-query trajectories m^Q is computed as follows,

$$m^Q = \frac{1}{n}\sum_{j=1}^{n}(R(\tau_j^Q)), \tag{39}$$

The advantage a_j^Q of the multi-query trajectory τ_j^Q is generated as follows,

$$a_j^Q = R(\tau_j^Q) - m^Q, \tag{40}$$

The multi-query symmetric loss $\bigtriangledown_\theta J^Q(\theta)$ of one instance based on all symmetric multi-query trajectories is as follows,

$$\bigtriangledown_\theta J^Q(\theta) = \frac{1}{n}\sum_{j=1}^{n} a_j^Q \bigtriangledown_\theta log P_j^Q, \tag{41}$$

where P_j^Q denotes the completely joint probability distribution of a multi-query trajectory τ_j^Q mentioned in Eq. (14).

Rotational Symmetric Loss. The rotation-symmetric loss is used to evaluate the performance of training with rotation-symmetric augmentation. Each augmented instance corresponds to a rotational trajectory similar to that in multi-query symmetric loss. PD-SNO computes the mean reward m^N of all rotational trajectories and the advantage a_j^N of each rotational trajectory after all trajectories finish rollout as follows,

$$m^N = \frac{1}{g}\sum_{j=1}^{g}(R(\tau_j^N)), \tag{42}$$

$$a_j^N = R(\tau_j^N) - m^N. \tag{43}$$

The rotation-symmetric loss $\bigtriangledown_\theta J^N(\theta)$ of one instance based on all rotational symmetric trajectories as follows,

$$\bigtriangledown_\theta J^N(\theta) = \frac{1}{n}\sum_{j=1}^{n} a_j^N \bigtriangledown_\theta log P_j^N, \tag{44}$$

Algorithm 1 PD-SNO Training

Input: epoch size o, batch size b, policy π_θ, the number of pick-up nodes/delivery nodes n, the number of augmentations g

1: **for** epoch = 1 to o **do**
2: **for** batch = 1 to b **do**
3: Generate training data h
4: **for** i = 1 to g **do**
5: $h_i \leftarrow \{h_1, h_2, ..., h_g\} \leftarrow$ rotation-symmetric augment (h)
6: **for** j = 1 to n **do**
7: $\tau_{ij}, P_{ij} \leftarrow$ multi-query symmetric rollout (h_i, π_θ)
8: **end for**
9: $a_{ij}^Q = (R(\tau_{ij}) - \frac{1}{n}\sum_{j=1}^{n} R(\tau_{ij}))$
10: $\bigtriangledown_\theta J_i^Q(\theta) = \frac{1}{n}\sum_{j=1}^{n} a_{ij}^Q \bigtriangledown_\theta log P_{ij}$
11: **end for**
12: $\bigtriangledown_\theta J^Q(\theta) = \frac{1}{g}\sum_{i=1}^{g} \bigtriangledown_\theta J_i^Q(\theta)$
13: $a_i^N, P_i \leftarrow (R(\tau_i) - \frac{1}{g}\sum_{i=1}^{g} R(\tau_i))$
14: $\bigtriangledown_\theta J^N(\theta) = \frac{1}{g}\sum_{i=1}^{g} a_i^N \bigtriangledown_\theta log P_i$
15: $\bigtriangledown_\theta J^B(\theta) = \alpha \bigtriangledown_\theta J^Q(\theta) + \beta \bigtriangledown_\theta J^N(\theta)$
16: $\theta \leftarrow \bigtriangledown_\theta J^B(\theta)$
17: **end for**
18: **end for**

where P_j^N denotes the completely joint probability distribution of a rotational trajectory τ_j^N.

Combined Loss. Both multi-query symmetric loss and rotation-symmetric loss used for efficiently training the neural network of PD-SNO, the combined loss $\bigtriangledown_\theta J^B(\theta)$ will generated as follows,

$$\bigtriangledown_\theta J^B(\theta) = \alpha \bigtriangledown_\theta J^Q(\theta) + \beta \bigtriangledown_\theta J^N(\theta) \tag{45}$$

where $\alpha + \beta = 1$ are the weights that determine the importance of both loss.

Training Algorithm. The training algorithm of our PD-SNO is displayed in Algorithm. 1. The details are as follows: Firstly, the training instances generated on the fly are shown in line 3. Specifically, the synthetic rules of training instances are display as follows: In PDTSP and m-PDTSP, all coordinates are real numbers and follow the uniform distribution in the square of [0, 1]; In m-PDTSP, all demands of pick-up nodes are positive integers and follow the uniform distribution in the range of [1, 10], where each demand of delivery node is the opposite number of the paired pick-up node. Then training instances are processed by the rotation-symmetric augmentation as training input and fed to the PD-SNO as line 5. After initiating the input, PD-SNO starts the multi-query symmetric rollout step by step and selects the node to visit according to the probability obtained from the policy, where each node has a chance to be selected except if they are masked by the problem constraints or have been visited. The rollout strategy deployed in training is the sampling rollout strategy. PD-SNO samples one solution in training for the neural network to better update the policy instead of sampling a large number of solutions to find the best one. After all multi-query trajectories finish the rollout, the PD-SNO cumulatively multiples the probability of each action (i.e., node selection) in the finished Hamiltonian cycle as P_{ij} as line 7. Then the multi-query symmetric loss is computed at line 10. Similarly, the rotation-symmetric loss is calculated as line 15. The above two loss functions are combined together to upgrade the policy θ as line 16.

4.4. Symmetric inference

Different from the primary goal of training, which is to update the policy, the primary goal of inference is to find the best solution with the best effort of the solver. The multi-query symmetric rollout, the

Algorithm 2 PD-SNO Inference

Input: evaluation data h, policy π_θ, the number of pick-up nodes/delivery nodes n, the number of augmentations g
Output: result
1: **for** i = 1 to g **do**
2: $h_i \leftarrow \{h_1, h_2, ..., h_g\} \leftarrow$ rotation-symmetric augment (h)
3: **for** j = 1 to n **do**
4: $\tau_{ij} \leftarrow$ multi-query symmetric rollout (h_i, π_θ)
5: **end for**
6: **end for**
7: result = argmax($\{\{R(\tau_{ij})\}_{i=1}^{g}\}_{j=1}^{n}$)

rotational symmetric augmentation, the exchanged symmetric augmentation, and either the greedy rollout strategy or the sampling rollout strategy are deployed during inference.

Exchanged Symmetric Augmentation Except for the multi-head symmetric attention mechanism, we further explore the symmetry between partitions of instances and propose an exchanged symmetric augmentation to effectively optimize the inference results. Specifically, we employ the exchanged symmetric augmentation to generate the new instance (depot, delivery nodes, pick-up nodes) based on the original instance (depot, pick-up nodes, delivery nodes) during the inference phase. Regardless of the PDTSP and the m-PDTSP, comprehensively exploring the symmetry of the partitions of instances is crucial for obtaining a better solution. Different from the coordinate rotation, the instance after exchanged symmetric augmentation is different from the original one in the view of either the policy network or humans. However, the solutions of those augmented instances can be used to solve the original instance after flipping the order, i.e., flipping the solutions $(A^0, A^1, A^2, ..., A^{2n+1})$ to $(A^{2n+1}, A^{2n}, A^{2n-1}, ..., A^0)$ that the solution not only meet the constraints including pick-up and delivery and capacity, but also expand the option of action and comprehensively be able to capture the relationship between partitions of specific instance. The instance augmentation deployed on the PD-SNO during inference including the coordinate rotation and the exchanged symmetric augmentation. To keep a reasonable amount of instances for inference, we only deploy the exchanged symmetric augmentation on the original instance rather than all 8 instances after coordinate rotation augmentation.

Inference Algorithm. The inference of the PD-SNO is displayed in Algorithm 2 and the details of the inference are introduced as follows: First, the evaluation instances are processed by instance augmentation and fed to PD-SNO as line 2. Then, the PD-SNO runs the multi-query symmetric rollout and generates g*n completely in the Hamiltonian cycle as line 4, where the rollout strategy can be greedy or sampling. Finally, the PD-SNO finds the best solution in the $g \times n$ generated Hamiltonian cycle to output as line 7.

5. Experiments result

In this section, we conducted a series of experiments to verify the following assumptions:

- The proposed PD-SNO is able to outperform various baselines on synthetic datasets, including traditional methods (both exact and heuristic algorithms) and DRL-based methods (both construction and improvement types).
- The proposed PD-SNO shows prominent generalization performance on benchmark datasets compared to various state-of-the-art DRL-based baselines (both construction and improvement types).
- The proposed symmetries of the PD-SNO, including encoder, decoder, rollout, training, and inference, contribute to the final performance, and the maximum performance is achieved when all symmetries are exploited simultaneously.

5.1. Experiment settings

The details of the experiment settings are introduced in this section, including baselines, hyper-parameters, and training. We use an 8-core AMD R7 CPU to evaluate the performance of traditional CPU-based baselines and a single RTX4090 GPU to evaluate that of DRL-based solvers, respectively. Our code are available online.[1]

Baselines. We compare our PD-SNO to various well-known baselines: **POMO** (Kwon et al., 2020), a state-of-the-art DRL-based construction method for solving TSP and CVRP. **Heter-AM** (Li et al., 2021), a state-of-the-art DRL-based construction method for solving PDTSP. **N2S** (Ma et al., 2022), a state-of-the-art DRL-based improvement method for solving PDTSP. **LKH3** (Helsgaun, 2017), a state-of-the-art heuristic solver for solving COPs. **Gurobi** (Achterberg, 2019), a fast exact solver for solving mathematical model-based optimization problems.

Hyper-parameters. The hyper-parameters of the PD-SNO are shown as follows: Regardless of the encoder or decoder, the dimension of embedding is set as $dim_E = 256$, the dimensions of query, key, and value are set as $dim_Q = 16$, and the number of heads in the MHSA mechanism is set as $Y = 16$; the number of encoder sub-layers is set as $U = 6$; the size of the tanh clip is set as CLIP = 10, and the dimensions of the feed-forward layer are set as $dim_F = 512$. We use the same optimizer hyper-parameter settings for all training. Specifically, we utilized the Adam optimizer (Kingma & Ba, 2014) with a learning rate $\eta = 10^{-4}$ and a decay of 0.1 at a milestone epoch for accelerating convergence. At last, we set a weight decay (L2 regularization) as $w = 10^{-6}$ to prevent the overfitting.

Training. Whether solving PDTSP or m-PDTSP, the number of training instances for the same DRL neural network is the same. The number of training epochs for all DRL-based methods is set at 2000. Thus, the number of instances in each epoch is adjusted to keep the number of total training instances sufficient for complete convergence. For each different DRL-based method, the number of instances in one epoch is shown as follows: **PD-SNO** Each epoch consists of 10,000 instances. **Heter-AM** Each epoch consists of 512,000 instances. **POMO** Each epoch consists of 100,000 instances. **N2S** Each epoch consists of 1200 instances.

5.2. Performance comparison

We first test the performance of our PD-SNO by comparing it with various well-known baselines on synthetic datasets. The details of settings, results on PDTSP, and results on m-PDTSP are displayed and discussed in this section. It is worth noting that our PD-SNO is one of the DRL-based construction methods, which can solve the NP-hard problems with a negligible running time compared to that in conventional optimization algorithms and DRL-based improvement algorithms. The reason is that the process of solving optimization problems by construction methods is similar as human's instinct, which directly decide every step completely without iteration. As a result, the different solving times in one construction method only mean sampling the final result from more finished solutions, not the trade-off between time spent and quality of solution. The reason we sample more solutions is to prove that our PD-SNO has ability to obtain the higher quality results.

Settings. The settings of the synthetic datasets are the same as the training ones. All methods were evaluated on the same sets of 2000 synthetic test instances for each problem size, including 21, 51, and 101. For a fair comparison, we explore the performance of each method under different solution times. Specifically, the different settings of solution time for traditional methods are shown as follows: We test

[1] https://github.com/Good9T/PD-SNO

Table 1
The comparison result on synthetic dataset of PDTSP.

Method	PDTSP-21			PDTSP-51			PDTSP-101		
	Obj.	Gap	Time (m)	Obj.	Gap	Time (m)	Obj.	Gap	Time (m)
Gurobi	4.550	**−0.02%**	2.62	–	–	–	–	–	–
Gurobi (120 s)	4.550	**−0.02%**	2.62	6.872	0.28%	1023.48	9.582	1.60%	4000.85
LKH3 (t.2000)	4.583	0.70%	14.84	6.893	0.58%	72.19	9.506	0.80%	346.18
LKH3 (t.5000)	4.551	0.00%	41.28	6.868	0.22%	160.89	9.468	0.39%	831.35
N2S (t.100)	4.563	0.26%	4.81	7.132	4.07%	6.68	10.213	8.29%	13.32
N2S (t.500)	4.554	0.07%	23.18	6.941	1.28%	34.41	9.695	2.80%	69.81
N2S (t.2000)	4.551	0.00%	102.35	6.865	0.18%	169.51	9.459	0.30%	342.18
Heter-AM (g.)	4.622	1.56%	0.02	7.312	6.70%	0.04	10.325	9.48%	0.08
Heter-AM (s.500)	4.596	0.99%	10.43	7.134	4.10%	21.68	10.148	7.60%	43.72
Heter-AM (s.4000)	4.573	0.48%	79.15	7.104	3.66%	189.26	10.062	6.69%	408.65
POMO (g.)	4.578	0.59%	0.03	7.120	3.90%	0.11	10.036	6.42%	0.21
POMO (s.500)	4.572	0.46%	15.19	7.046	2.82%	55.69	9.834	4.27%	106.35
POMO (s.2000)	4.569	0.40%	25.64	7.018	2.41%	108.67	9.793	3.84%	413.84
PD-SNO (g.)	4.568	0.37%	0.09	6.901	0.70%	0.34	9.542	1.18%	0.67
PD-SNO (s.100)	4.562	0.24%	3.95	6.862	0.13%	18.91	9.463	0.34%	67.42
PD-SNO (s.500)	4.551	[a]0.00%	20.03	6.853	[a]**0.00%**	96.32	9.431	[a]**0.00%**	341.25

The gap with boldface indicates that the corresponding method is the best among all methods.
[a] Indicates that the corresponding method is the best among DRL-based methods.

Table 2
The comparison result on synthetic dataset of m-PDTSP.

Method	m-PDTSP-21			m-PDTSP-51			m-PDTSP-101		
	Obj.	Gap	Time (m)	Obj.	Gap	Time (m)	Obj.	Gap	Time (m)
Gurobi	5.015	**−0.16%**	2.67	–	–	–	–	–	–
Gurobi (120 s)	5.015	**−0.16%**	2.67	9.312	0.41%	1143.32	16.144	1.70%	4001.23
LKH3 (t.2000)	5.028	0.10%	16.45	9.341	0.72%	74.89	15.995	0.76%	362.48
LKH3 (t.5000)	5.025	0.04%	43.86	9.312	0.41%	164.77	15.946	0.45%	905.14
N2S (t.100)	5.043	0.40%	5.23	9.602	3.54%	7.21	17.289	8.91%	14.82
N2S (t.500)	5.026	0.06%	27.15	9.425	1.63%	38.11	16.486	3.86%	76.41
N2S (t.2000)	5.023	0.00%	118.32	9.309	0.38%	174.93	16.082	1.31%	359.78
Heter-AM (g.)	5.124	2.01%	0.02	9.775	5.40%	0.04	17.386	9.53%	0.08
Heter-AM (s.500)	5.081	1.15%	11.07	9.684	4.42%	24.93	17.132	7.92%	56.14
Heter-AM (s.4000)	5.049	0.52%	83.42	9.612	3.64%	196.19	16.832	6.04%	412.61
POMO (g.)	5.056	0.66%	0.03	9.609	3.61%	0.12	16.798	5.82%	0.23
POMO (s.500)	5.049	0.52%	8.02	9.553	3.01%	29.41	16.541	4.20%	113.79
POMO (s.2000)	5.043	0.40%	27.86	9.513	2.58%	112.55	16.428	3.49%	431.48
PD-SNO (g.)	5.041	0.36%	0.04	9.505	2.49%	0.19	16.155	1.77%	0.74
PD-SNO (s.100)	5.036	0.26%	4.12	9.362	0.95%	19.32	15.992	0.74%	71.2
PD-SNO (s.500)	5.023	[a]0.00%	20.25	9.274	[a]**0.00%**	98.93	15.874	[a]**0.00%**	352.01

The gap with boldface indicates that the corresponding method is the best among all methods.
[a] Indicates that the corresponding method is the best among DRL-based methods.

the performance of **Gurobi** under a 120-second time limit (tl.120 s) and without any time limit separately. It is worth noting that Gurobi cannot obtain each solution of PDTSP-51, PDTSP-101, m-PDTSP-51 and m-PDTSP-101 in an acceptable time, i.e., no solution may be found even after a 10-minute period on instances of PDTSP-51. Thus, we only record the performance of Gurobi on PDTSP-21 without time limits. We test the performance of **LKH3** under the step limits of 2000 (t.2000) and 5000 (t.5000), separately. However, the solution times of DRL-based methods can be easily distinguished by the different instance augmentations and rollout strategies. For all DRL-based construction methods, we use both coordinates augmentation and exchange augmentation; for N2S, we use the instance augmentation original proposed in Ma et al. (2022). The different settings of solution time are shown as follows: We separately test the performance of **PD-SNO** through greedy rollout strategy (g.) and sampling rollout strategy (s.), where the sampling strategies include sampling 100 solutions (s.100) and sampling 500 solutions (s.500); we separately test the performance of **POMO** through POMO (g.), POMO (s.500), and POMO (s.2000); we separately test the performance of **Heter-AM** through Heter-AM (g.), Heter-AM (s.500) and Heter-AM (s.4000); we separately test the performance of **N2S** under the step limits of 100 (t.100), 500 (t.500), and 2000 (t.2000).

Results on PDTSP. Table 1 reports the comparison results on PDTSP, where the objective value, gap to PD-SNO (s.500), and the total solving time in minutes are recorded and shortened as the Obj., Gap, and Time (m), respectively. As the results show, DRL-based construction methods show an overwhelming advantage in total solving time when used with the greedy rollout strategy. Against the POMO and Heter-AM, we compare the PD-SNO (g.) with the POMO (g.) and Heter-AM (g.) to testify that the PD-SNO outperforms the POMO and Heter-AM when all of them are using the greedy rollout strategy; we compare the PD-SNO (s.500) to the POMO (s.500) and Heter-AM (s.500) to testify that the PD-SNO outperforms the POMO and Heter-AM while they sampling same number of solutions (i.e., sampling 500 solutions); we compare the PD-SNO (s.500) to the POMO (s.2000) and Heter-AM (s.4000) to testify that the PD-SNO outperforms the POMO and Heter-AM while all of them using the sampling strategy and under a comparable total solving time on PDTSP-101 (i.e., 341.25 m, 413.84 m, 408.65 m). Against the state-of-the-art DRL-based baseline N2S, we compare the PD-SNO (g.), PD-SNO (s.100), and PD-SNO (s.500) with the N2S (t.100), N2S (t.500), and N2S (t.2000) to specifically testify the performance. We compare the PD-SNO (s.100) to the N2S (t.100) and compare the PD-SNO (s.500) to the N2S (t.500) to testify that the PD-SNO outperforms the N2S while sampling same number of solutions; we compare the PD-SNO (g.) to the N2S (t.100) to testify that the PD-SNO overwhelmingly outperforms the N2S while they are under a restricted solution time limits; we compare the PD-SNO (s.100) to the N2S (t.500) to testify that the PD-SNO still outperforms the N2S while loosening the time limit by 5 times. Thus, the most prominent advantage of DRL-based

construction methods over DRL-based improvement methods is that they obtain a decent solution at an exquisitely low computational cost. Moreover, we further compare the PD-SNO (s.500) to the N2S (t.2000) to testify that the PD-SNO is still comparable to the N2S and gains a slight advantage while further loosening the time limit by 5 times. The PD-SNO is slightly inferior to the Gurobi on the PDTSP-21 due to the suitability of the exact solver for solving small-scale problems. However, the PD-SNO still outperforms the Gurobi (tl.120 s) on PDTSP-51 and PDTSP-101. Compared to the state-of-the-art heuristic solver LKH3, the PD-SNO (s.500) outperforms the LKH3 (t.5000) on PDTSP-51 and PDTSP-101 with gaps of 0.22% and 0.39%.

Results on m-PDTSP. Table 2 reports the comparison results on m-PDTSP, where the objective value, gap to PD-SNO (s.500), and the total solving time in minutes are recorded and shortened as the Obj., Gap, and Time (m), separately. As the results display, DRL-based construction methods still show an overwhelming advantage in total solving time. Against the DRL-based construction baselines, our PD-SNO outperforms all of them while simultaneously using either a greedy strategy or a sampling strategy with a comparable sampling number of solutions or total solving time, which is similar as in PDTSP. Against the state-of-the-art DRL-based improvement baselines, the results of PD-SNO (g.), PD-SNO (s.100), and PD-SNO (s.500) are compared to the N2S (t.100), N2S (t.500), and N2S (t.2000) to specifically testify the performance. The PD-SNO (s.100) is compared to the N2S (t.100) and the PD-SNO (s.500) is compared to the N2S (t.500) to testify that the PD-SNO outperforms the N2S while sampling same number of solutions; the PD-SNO (g.) is compared the N2S (t.100) to testify that the PD-SNO overwhelmingly outperforms the N2S while they are under the most restricted solution time limits; the PD-SNO (s.100) is compared to the N2S (t.500) to testify that the PD-SNO still outperforms the N2S while loosening the time limit; the PD-SNO (s.500) is compared to the N2S (t.2000) to testify that the PD-SNO is still comparable to the N2S and gains a slight advantage without time limit. The N2S subject to the capacity constraint in m-PDTSP, where the gaps between the N2S to our PD-SNO is larger than in PDTSP, i.e., the gap in the PDTSP-101 is 0.30% while the gap in the m-PDTSP-101 is 1.31%. The PD-SNO is slightly inferior to the Gurobi on the m-PDTSP-21 due to the suitability of the exact solver for solving small-scale problems. However, the PD-SNO still outperforms the Gurobi (tl.120 s) on PDTSP-51 and PDTSP-101. Compared to the state-of-the-art traditional solver LKH3, the PD-SNO (s.500) outperforms the LKH3 (t.5000) on PDTSP-21, PDTSP-51, and PDTSP-101 with gaps of 0.04%, 0.41%, and 0.45%.

5.3. Generalization experiments

We then test the generalization performance of our PD-SNO on benchmark datasets obtained from (Renaud, Boctor, & Laporte, 2002), and compare the performance with various well-known DRL-based baselines. In the deep learning domain, generalization performance is a key indicator to evaluate whether a model can be adapted to real-world problems and effectively solve them. The details of settings, results on PDTSP, and results on m-PDTSP are displayed and discussed in this section.

Settings. In the generalization experiments, we uniformly use the DRL-based methods that were trained on PDTSP-101 synthetic datasets to solve PDTSP-101 and PDTSP-201 instances on benchmark datasets. Therefore, we compare our PD-SNO (s.500) with the POMO (s.2000) and N2S (t.2000) to test the relative generalization performance while using a comparable total solving time. The total solving times of all DRL-based baselines used in the generalization experiments are comparable, which can be inferred from Table 1. The problem size of the first 10 instances in the benchmark dataset is the same as the training dataset, so all DRL-based methods can yield a better solution on them than on the instances with other problem sizes. However, the problem size of the last 10 instances has a large shift when compared to the training dataset, and those instances are definitely hard to optimize for DRL-based methods. All methods tested these instances 10 times, and the best solution for each instance was yielded by the PD-SNO after testing 10 times, and we record the best objective value and the average objective value of them. The instance augmentations used in the generalization experiments are the same as in Section 5.2.

Results on PDTSP. Table 3 reports the results of generalization experiments, including the index of instances as 'Ins.', the problem size, the best objective value as 'B.Obj', the average objective value as 'A.Obj', the gap of the best objective value to the best objective value of PD-SNO (s.500) as 'B.Gap', and the gap of the average value of 20 solutions to the best objective value of PD-SNO (s.500) as 'A.Gap'. The best solution for each instance was yielded by the PD-SNO in this experiment. In 10 PDTSP-101 instances, the best solution for each instance was yielded by the PD-SNO in this experiment, where the average 'B.Gap' of the N2S is (0.30%) and the POMO is (1.29%); the average gap among 10 instances of the PD-SNO (0.31%) is the lowest in this experiment, where the 'A.Gap' of the N2S (0.63%) and the POMO is (2.19%). In 10 PDTSP-201 instances, the best solution for each instance was yielded by the PD-SNO in this experiment, where the average 'B.Gap' of the N2S is (0.83%) and the POMO is (3.62%); the average gap among 10 larger-scale instances of the PD-SNO (1.03%) is lower than the gap of the N2S (1.48%) and the POMO (6.94%).

Results on m-PDTSP. Table 4 reports the results of generalization experiments, including the index of instances as 'Ins.', the problem size, the best objective value as 'B.Obj', the average objective value as 'A.Obj', the gap of the best objective value to the best objective value of PD-SNO (s.500) as 'B.Gap', and the gap of the average value to the best objective value of PD-SNO (s.500) as 'A.Gap'. Our PD-SNO (s.500) outperforms the state-of-the-art DRL-based construction baseline POMO (s.2000) and DRL-based improvement baseline N2S (t.2000) in all instances, regardless of whether the problem size is 101 or 201. In 10 m-PDTSP-101 instances, the best solution for each instance was yielded by the PD-SNO in this experiment, where the average 'B.Gap' of the N2S is (1.28%) and the POMO is (0.92%); the average gap among 10 instances of the PD-SNO (0.29%) is the lowest in this experiment, where the 'A.Gap' of the N2S (2.14%) and the POMO is (1.90%). In 10 m-PDTSP-201 instances, the best solution for each instance was yielded by the PD-SNO in this experiment, where the average 'B.Gap' of the N2S is (1.72%) and the POMO is (2.74%); the average gap among 10 larger-scale instances of the PD-SNO (0.84%) is lower than the gap of the N2S (2.81%) and the POMO (6.45%). The N2S subject to the capacity constraint in m-PDTSP, where the 'A.Gap' between the N2S to our PD-SNO is larger than in PDTSP, i.e., 0.63% in the PDTSP-101 while 2.14% in the m-PDTSP-101.

5.4. Ablation studies

To further highlight each proposed design, including the encoder, decoder, and training, we conduct ablation studies by gradually removing the proposed designs from the PD-SNO and replacing them with components from the state-of-the-art DRL-based construction method POMO. We use 'group' to denote a combination of the designs.

Settings. We train the DRL network of each group on PDTSP-101 and m-PDTSP-101 synthetic training dataset, separately. Then the DRL network of each group is evaluated on 2000 PDTSP-101 synthetic test instances. Besides, to maintain the consistency of the ablation studies, we uniformly deploy the strategy of sampling 500 solutions as the rollout strategy during inference and the coordinate rotation as the instance augmentation.

Results. Table 5 reports the objective value as 'Obj.' and the gap to the PD-SNO (deployed with all proposed designs), i.e., group 1. We use '–' to denote the default design from the PD-SNO and 'POMO' to denote the design from the state-of-the-art DRL-based construction solver POMO.

Table 3
The generalization result on benchmark dataset of PDTSP.

Ins.	Size	PD-SNO (s.500)			N2S (t.2000)			POMO (s.2000)		
		B.Obj	A.Obj.	A.Gap	B.Gap	A.Obj.	A.Gap	B.Gap	A.Obj.	A. Gap
P01	101	**799**	801	**0.25%**	0.16%	802	0.38%	1.78%	828	3.63%
P02	101	**729**	733	**0.55%**	0.26%	732	0.41%	1.02%	741	1.65%
P03	101	**748**	749	**0.13%**	0.12%	751	0.40%	0.68%	756	1.07%
P04	101	**807**	808	**0.12%**	0.85%	819	1.49%	1.05%	819	1.49%
P05	101	**783**	786	**0.38%**	0.22%	785	0.26%	1.54%	801	2.30%
P06	101	**755**	757	**0.26%**	0.17%	758	0.40%	1.16%	769	1.85%
P07	101	**767**	769	**0.26%**	0.65%	774	0.91%	1.31%	787	2.61%
P08	101	**762**	765	**0.39%**	0.15%	764	0.26%	1.22%	783	2.76%
P09	101	**766**	768	**0.26%**	0.13%	767	0.13%	1.08%	776	1.31%
P10	101	**754**	758	**0.53%**	0.32%	766	1.59%	2.01%	778	3.18%
A.		**767**	769	**0.31%**	0.30%	772	0.63%	1.29%	784	2.19%
P11	201	**1059**	1071	**1.13%**	0.26%	1068	0.85%	2.53%	1113	5.10%
P12	201	**1096**	1106	**0.91%**	1.06%	1127	2.83%	1.65%	1122	2.37%
P13	201	**1092**	1101	**0.82%**	1.06%	1112	1.83%	3.02%	1206	10.44%
P14	201	**1066**	1073	**0.66%**	0.63%	1076	0.94%	1.76%	1121	5.16%
P15	201	**1078**	1092	**1.30%**	1.04%	1094	1.48%	4.63%	1194	10.76%
P16	201	**1073**	1084	**1.03%**	0.86%	1086	1.21%	2.12%	1128	5.13%
P17	201	**1068**	1079	**1.03%**	0.62%	1076	0.75%	4.25%	1171	9.64%
P18	201	**1097**	1110	**1.19%**	0.74%	1112	1.37%	5.93%	1134	3.37%
P19	201	**1066**	1076	**0.94%**	1.02%	1086	1.88%	6.42%	1166	9.38%
P20	201	**1108**	1122	**1.26%**	1.05%	1126	1.62%	3.86%	1198	8.12%
A.		**1080**	1091	**1.03%**	0.83%	1096	1.48%	3.62%	1155	6.94%

The gap with boldface indicates that the corresponding method is the best among DRL-based methods.

Table 4
The generalization result on benchmark dataset of m-PDTSP.

Ins.	Size	PD-SNO (s.500)			N2S (t.2000)			POMO (s.2000)		
		B.Obj	A.Obj.	A.Gap	B.Gap	A.Obj.	A.Gap	B.Gap	A.Obj.	A. Gap
P01	101	**1649**	1653	**0.24%**	1.21%	1682	2.00%	0.46%	1668	1.15%
P02	101	**1741**	1743	**0.11%**	1.32%	1784	2.47%	0.38%	1757	0.92%
P03	101	**1710**	1721	**0.64%**	1.51%	1771	3.57%	0.95%	1736	1.52%
P04	101	**1739**	1749	**0.58%**	1.46%	1771	1.84%	0.64%	1772	1.90%
P05	101	**1630**	1632	**0.12%**	1.62%	1668	2.33%	0.54%	1649	1.17%
P06	101	**1754**	1756	**0.11%**	0.92%	1786	1.82%	1.02%	1785	1.77%
P07	101	**1558**	1563	**0.32%**	1.03%	1584	1.67%	1.31%	1591	2.12%
P08	101	**1756**	1759	**0.17%**	1.56%	1797	2.33%	1.29%	1802	2.62%
P09	101	**1840**	1847	**0.38%**	1.25%	1876	1.96%	1.15%	1886	2.50%
P10	101	**1643**	1646	**0.18%**	0.93%	1665	1.34%	1.48%	1697	3.29%
A.		**1702**	1707	**0.29%**	1.28%	1738	2.14%	0.92%	1734	1.90%
P11	201	**2942**	2966	**0.82%**	1.18%	3019	2.62%	2.53%	3121	6.08%
P12	201	**2998**	3031	**1.10%**	1.81%	3102	3.47%	2.39%	3185	6.24%
P13	201	**3055**	3079	**0.79%**	1.92%	3138	2.72%	3.02%	3224	5.53%
P14	201	**2963**	2989	**0.88%**	1.94%	3054	3.07%	1.48%	3162	6.72%
P15	201	**3177**	3198	**0.66%**	1.97%	3264	2.74%	2.92%	3349	5.41%
P16	201	**3246**	3274	**0.86%**	1.71%	3326	2.46%	3.78%	3461	6.62%
P17	201	**3241**	3264	**0.71%**	1.82%	3334	2.87%	2.56%	3484	7.50%
P18	201	**3438**	3464	**0.76%**	1.23%	3529	2.65%	2.73%	3589	4.39%
P19	201	**3366**	3398	**0.95%**	1.92%	3466	2.97%	2.89%	3625	7.69%
P20	201	**3297**	3328	**0.94%**	1.67%	3382	2.58%	3.05%	3569	8.25%
A.		**3172**	3199	**0.84%**	1.72%	3261	2.81%	2.74%	3377	6.45%

The gap with boldface indicates that the corresponding method is the best among DRL-based methods.

Table 5
The ablation studies.

Group	Design				PDTSP-101		m-PDTSP-101	
	Encoder	Decoder	Training	Augment	Obj.	Gap	Obj.	Gap
1	–	–	–	–	9.431	0.00%	15.874	0.00%
2	POMO	–	–	–	9.562	1.39%	16.088	1.35%
3	–	POMO	–	–	9.574	1.52%	16.106	1.46%
4	–	–	POMO	–	9.584	1.62%	16.102	1.44%
5	–	–	–	POMO	9.497	0.70%	16.002	0.81%
6	POMO	POMO	POMO	POMO	9.834	4.27%	16.541	4.20%

For comparing the designs of encoders, the gap between group 1 and group 2 shows that the proposed encoder deployed with a multi-head symmetric attention can improve the solutions by 1.39% on PDTSP and 1.35% on m-PDTSP; for comparing the designs of decoders, the gap between group 1 and group 3 shows that the proposed decoder deployed with a multi-head symmetric attention can also improve the solutions by 1.52% on PDTSP and 1.46% on m-PDTSP; for comparing the designs of training algorithms, the gap between group 1 and group 4 shows that the proposed training algorithm deployed with two symmetric loss functions can improve the solutions by 1.62% on PDTSP and 1.44%

on m-PDTSP; for comparing the designs of instance augmentation, the gap between group 1 and group 5 shows that additionally deploying the proposed exchanged symmetric augmentation can improve the solutions by 0.70% on PDTSP and 0.81% on m-PDTSP based on the solutions only deployed a coordinates rotation augmentation. As a conclusion, each design in our PD-SNO can improve the solutions independently, and the most prominent performance can be achieved while all proposed designs are deployed simultaneously.

6. Conclusion and future work

In this paper, a symmetric neural optimization on deep reinforcement learning-based construction method has been proposed to deal with PDPs, i.e., PD-SNO. The problems solved by the PD-SNO include PDTSP and m-PDTSP. In order to capture the relationship between symmetric partitions of instances, a multi-head symmetric attention (MHSA) mechanism has been designed and deployed in the encoder; in order to construct the solutions by gathering the symmetric node embeddings and the constantly updated environment, the MHSA is modified and deployed in the decoder; in order to train the policy network by leveraging the symmetries between different solution trajectories, a loss function based on the multi-query rollout and instance augmentation has been proposed; in order to yield a better solution during the inference phase, an exchanged symmetric augmentation has been designed.

In the experiments, the proposed PD-SNO has been compared with various state-of-the-art DRL-based baselines and a fastest exact solver on both synthetic datasets and benchmark datasets. The experimental results show that the proposed PD-SNO outperforms other DRL-based baselines regardless of the performance and the generalization performance. The effectiveness of each design in the PD-SNO further testified through the ablation studies.

In the future, the proposed PD-SNO will be used to investigate PDP with more variants, such as PDTSP with a time window and one-commodity PDTSP to show its performance on them. Besides, the applications in the real-world transportation and logistics usually contain multiple objective, thus the idea of PD-SNO will be extended to solve multi-objective problems via deep reinforcement learning.

CRediT authorship contribution statement

Jinqi Li: Conceptualization, Methodology, Software, Writing – original draft. **Yunyun Niu:** Conceptualization, Methodology, Supervision, Funding acquisition. **Guodong Zhu:** Conceptualization, Methodology, Writing – review & editing. **Jianhua Xiao:** Conceptualization, Methodology, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The code of PD-SNO and the dataset used in our study can be found at https://github.com/Good9T/PD-SNO.

Acknowledgments

This research was supported by the National Natural Science Foundation of China (62172373, 61872325, 62072258); the Fundamental Research Funds for the Central Universities, China (2652019028).

References

Achterberg, T. (2019). What's new in gurobi 9.0. Webinar Talk url: https://www.gurobi.com/wp-content/uploads/2019/12/Gurobi-90-Overview-Webinar-Slides-1.pdf.

Bellman, R. (1957). A Markovian decision process. *Journal of Mathematics and Mechanics*, 679–684.

Berbeglia Gerardo, G. I., & Gilbert, L. (2007). Static pickup and delivery problems: a classification scheme and survey. *TOP*, *15*, 1–31.

Çağrı Koç, Laporte, G., & Tükenmez, İ. (2020). A review of vehicle routing with simultaneous pickup and delivery. *Computers & Operations Research*, *122*, Article 104987. http://dx.doi.org/10.1016/j.cor.2020.104987, URL https://www.sciencedirect.com/science/article/pii/S0305054820301040.

Cook, W., Lovász, L., Seymour, P. D., et al. (1995). *vol. 20, Combinatorial optimization: papers from the DIMACS Special Year*. American Mathematical Soc..

Dubey, N., & Tanksale, A. (2023). A multi-depot vehicle routing problem with time windows, split pickup and split delivery for surplus food recovery and redistribution. *Expert Systems with Applications*, *232*, Article 120807. http://dx.doi.org/10.1016/j.eswa.2023.120807, URL https://www.sciencedirect.com/science/article/pii/S095741742301309X.

Hang Xu, S. R., & Arunapuram, S. (2003). Solving a practical pickup and delivery problem. *European Journal of Operational Research*, *37*(3), 347–364.

Helsgaun, K. (2017). *12, An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems*. Roskilde: Roskilde University.

Hernández-Pérez, H., & Salazar-González, J.-J. (2009). The multi-commodity one-to-one pickup-and-delivery traveling salesman problem. *European Journal of Operational Research*, *196*(3), 987–995.

Kakade, S. M. (2001). A natural policy gradient. *Advances in Neural Information Processing Systems*, *14*.

Kalatzantonakis, P., Sifaleras, A., & Samaras, N. (2023). A reinforcement learning-variable neighborhood search method for the capacitated vehicle routing problem. *Expert Systems with Applications*, *213*, Article 118812. http://dx.doi.org/10.1016/j.eswa.2022.118812, URL https://www.sciencedirect.com/science/article/pii/S0957417422018309.

Kim, M., Park, J., & Park, J. (2022). Sym-nco: Leveraging symmetricity for neural combinatorial optimization. *Advances in Neural Information Processing Systems*, *35*, 1936–1949.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Kool, W., Van Hoof, H., & Welling, M. (2018). Attention, learn to solve routing problems!. arXiv preprint arXiv:1803.08475.

Kwon, Y.-D., Choo, J., Kim, B., Yoon, I., Gwon, Y., & Min, S. (2020). Pomo: Policy optimization with multiple optima for reinforcement learning. *Advances in Neural Information Processing Systems*, *33*, 21188–21198.

Li, C., Gong, L., Luo, Z., & Lim, A. (2019). A branch-and-price-and-cut algorithm for a pickup and delivery problem in retailing. *Omega*, *89*, 71–91.

Li, J., Xin, L., Cao, Z., Lim, A., Song, W., & Zhang, J. (2021). Heterogeneous attentions for solving pickup and delivery problem via deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, *23*(3), 2306–2315.

Ma, Y., Li, J., Cao, Z., Song, W., Guo, H., Gong, Y., et al. (2022). Efficient neural neighborhood search for pickup and delivery problems. arXiv preprint arXiv:2204.11399.

Mosheiov, G. (1994). The Travelling Salesman Problem with pick-up and delivery. *European Journal of Operational Research*, *79*(2), 299–310. http://dx.doi.org/10.1016/0377-2217(94)90360-3.

Öztaş, T., & Tuş, A. (2022). A hybrid metaheuristic algorithm based on iterated local search for vehicle routing problem with simultaneous pickup and delivery. *Expert Systems with Applications*, *202*, Article 117401. http://dx.doi.org/10.1016/j.eswa.2022.117401, URL https://www.sciencedirect.com/science/article/pii/S095741742200745X.

Parragh Sophie, N., Doerner Karl, F., & Hartl Richard, F. (2008). A survey on pickup and delivery problems. *J. Betriebswirtschaft*, *58*, 21–51.

Renaud, J., Boctor, F. F., & Laporte, G. (2002). Perturbation heuristics for the pickup and delivery traveling salesman problem. *Computers & Operations Research*, *29*(9), 1129–1141.

Rodríguez-Martín, I., & Salazar-González, J. J. (2011). The multi-commodity one-to-one pickup-and-delivery traveling salesman problem: A matheuristic. In J. Pahl, T. Reiners, & S. Voß (Eds.), *Network optimization* (pp. 401–405). Springer Berlin Heidelberg.

Ruland, K., & Rodin, E. (1997). The pickup and delivery problem: Faces and branch-and-cut algorithm. *Computers & Mathematics with Applications*, *33*(12), 1–13. http://dx.doi.org/10.1016/S0898-1221(97)00090-4, URL https://www.sciencedirect.com/science/article/pii/S0898122197000904.

Savelsbergh, M. W. P., & Sol, M. (1995). The general pickup and delivery problem. *Transportation Science*, *29*(1), 1–105.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., et al. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, *30*.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, *8*(3), 229–256.

Wu, Y., Fan, M., Cao, Z., Gao, R., Hou, Y., & Sartoretti, G. (2023). Collaborative deep reinforcement learning for solving multi-objective vehicle routing problems. In *23rd international conference on autonomous agents and multi-agent systems*. 23rd International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2024; Conference date: 06-05-2024 Through 10-05-2024.

Yıldız, E. A., Karaoğlan, İ., & Altiparmak, F. (2023). An exact algorithm for two-echelon location-routing problem with simultaneous pickup and delivery. *Expert Systems with Applications*, *231*, Article 120598. http://dx.doi.org/10.1016/j.eswa.2023.120598, URL https://www.sciencedirect.com/science/article/pii/S0957417423011004.

Yu, V. F., Aloina, G., Jodiawan, P., Gunawan, A., & Huang, T.-C. (2023). The vehicle routing problem with simultaneous pickup and delivery and occasional drivers. *Expert Systems with Applications*, *214*, Article 119118. http://dx.doi.org/10.1016/j.eswa.2022.119118, URL https://www.sciencedirect.com/science/article/pii/S0957417422021364.