Information Sciences 619 (2023) 930–946

Contents lists available at ScienceDirect

Information Sciences

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

Solving combinatorial optimization problems over graphs with BERT-Based Deep Reinforcement Learning

Qi Wang [a,*], Kenneth H. Lai [b], Chunlei Tang [c,d]

[a] Shanghai Key Laboratory of Data Science, School of Computer Science, Fudan University, Shanghai, China
[b] Department of Computer Science, Brandeis University, Waltham, MA 02453, USA
[c] Harvard Medical School, Boston, MA 02120, USA
[d] Institute for Data Industry, Fudan University, Shanghai, China

ARTICLE INFO

Article history:
Received 1 October 2020
Received in revised form 12 November 2022
Accepted 14 November 2022
Available online 21 November 2022

Keywords:
Combinatorial optimization
BERT
Deep reinforcement learning

ABSTRACT

Combinatorial optimization, such as vehicle routing and traveling salesman problems for graphs, is NP-hard and has been studied for decades. Many methods have been proposed for its possible solution, including, but not limited to, exact algorithms, approximate algorithms, heuristic algorithms, and solution solvers. However, these methods cannot learn the problem's internal structure nor generalize to similar or larger-scale problems. Recently, deep reinforcement learning has been applied to combinatorial optimization and has achieved convincing results. Nevertheless, the challenge of effective integration and training improvement still exists. In this study, we propose a novel framework (BDRL) that combines BERT (Bidirectional Encoder Representations from Transformers) and deep reinforcement learning to tackle combinatorial optimization over graphs by treating general optimization problems as data points under an identified data distribution. We first improved the transformer encoder of BERT to embed the combinatorial optimization graph effectively. By employing contrastive objectives, we extend BERT-like training to reinforcement learning and acquire self-attention-consistent representations. Next, we used hierarchical reinforcement learning to pre-train our model; that is, to train and fine-tune the model through an iterative process to make it more suitable for a specific combinatorial optimization problem. The results demonstrate our proposed framework's generalization ability, efficiency, and effectiveness in multiple tasks.

1. Introduction

Combinatorial optimization (CO) [9] problems on graphs are a class of integer-constrained optimization problems and NP-hard problems, such as the representative traveling salesman problem (TSP) [31] and vehicle routing problem (VRP) [29,33,45], which are difficult to solve in polynomial time. Over the years, many classic algorithms have been proposed to solve CO problems on graphs. Examples include exact algorithms guaranteed to find optimal solutions; approximate algorithms that find approximate solutions; heuristic algorithms [37] such as ant colony optimization, genetic algorithms, and simulated annealing [30,24]; solution solvers; and others. These algorithms can be used alone or with their respective advantages to forming more robust algorithms [46]. However, even state-of-the-art traditional decision-making methods

* Corresponding author.
E-mail addresses: 17110240039@fudan.edu.cn (Q. Wang), klai5@bwh.harvard.edu (K.H. Lai), towne@fudan.edu.cn (C. Tang).

https://doi.org/10.1016/j.ins.2022.11.073
0020-0255/

require too much calculation time or are not mathematically well-defined. Moreover, they cannot learn the problem's internal structure, so they cannot generalize to larger-scale or similar problems. Most CO problems of the same type on a graph have similar structures, the difference between them generally being only the data and variables.

Deep learning and reinforcement learning (RL) have recently been used to develop practical solutions to combinatorial optimization problems [38]. CO problems on graphs belong to the class of sequential decision problems, in which deep learning and reinforcement learning have been used for approximation and reasoning. Neural network architectures for translation (e.g., pointer networks [35], transformers [34], etc.) can remember and store sequence data as well as mappings from one sequence to another. When applying deep learning to CO problems, representation learning is also necessary when preprocessing graph data. RL, which focuses on how agents learn from their interactions with the environment, is applied to optimize sequential decision problems. Deep reinforcement learning [38], which uses deep learning to approximate components of RL, has led to a revolutionary breakthrough in artificial intelligence.

However, there are specific difficulties and challenges in applying deep learning and reinforcement learning in combinatorial optimization. For example, network architectures built on recurrent neural networks (RNNs) may lead to limited parallel computing, low computing efficiency, and long-distance dependency problems. Traditional policy gradient-based RL algorithms (such as REINFORCE [38]) usually have low sampling efficiency. Since they require many Monte Carlo rollouts to obtain numerous trajectories with positive terminal rewards, especially in the early stages of learning and when the reward signals are sparse. Value-based algorithms such as Q-learning [38] also have the problem of inadequate exploration space. It isn't easy to develop simple, effective RL agents that depend only on raw high-dimensional inputs. In particular, it frequently calls for training several neural networks at once using sparse environment feedback and a high correlation between related observations. Due to the frequent noisy gradients in RL, this is especially challenging when the networks are large and densely connected, as in the case of transformers. Regarding generalization ability, models based on an encoder-decoder structure need a complex decoding process to handle different problems.

To meet the above challenges, we propose a BERT-based neural network framework, BDRL (BERT-Based Deep Reinforcement Learning), for combinatorial optimization. BERT [8] gives up the recurrent structure of RNNs, relying instead on position coding to assist the model in describing the spatiotemporal correlation information in the sequence data. Therefore, it can take care of the entire input sequence simultaneously, with a higher degree of parallelization and faster calculation speed. Graph-BERT (and Graph-Transformer) [47], a graph neural network [38] that learns graph representations solely based on the attention mechanism without relying on convolution-like or aggregation-like operations, does not consider connection information between nodes. Transformers have been successfully applied to the RL domain, given the similarities between sequential data processing in language modeling and reinforcement learning. As inspiration for GTrXL, Parisotto et al. [32] stated that extra gating was beneficial to train transformers for reinforcement learning given the high variance of the gradients in reinforcement learning, roughly equivalent to (un)supervised learning problems. We draw inspiration for representation learning from earlier research demonstrating how contrastive objectives enhance agent performance [11,22]. In particular, we combine the representation learning through the Invariant Causal Mechanisms [28] contrastive method and the masked prediction paradigm from BERT. Since reinforcement learning problems lack distinct objectives in contrast to language tasks, extending the BERT-masked prediction to reinforcement learning is difficult. We employ RELIC [28] in the time domain as a proxy supervision signal for the masked prediction to get around this problem. For the agent to successfully incorporate previously seen knowledge in the transformer weights, such a signal seeks to acquire self-attention-consistent representations containing the necessary information.

We define the VRP problem as a Markov decision process to apply reinforcement learning to optimize the model. We construct our framework entirely on the attention mechanism and further improve the encoder in the transformer by applying ReZero [1] to improve the graph embedding to get better contextual information from the nodes in the graph. We use bidirectional mask prediction, combined with generalization from contrastive learning, to learn better representations for transformers in RL without manual data augmentation. We employ hierarchical reinforcement learning to pre-train BDRL and fine-tune it to accommodate different combinatorial optimization problems.

To summarize, the main contributions of this paper are as follows:

- We are the first to consider the application of BERT to combinatorial optimization. Compared with previous encoder-decoder frameworks, BDRL does not need to redesign the decoder for different problems but can fine-tune the model for specific tasks.
- We have designed a novel transformer encoder layer in conjunction with ReZero and improved multi-head attention to improving our graph embeddings for contextual information from nodes.
- We design a novel contrastive representation learning objective that incorporates mask predictions from BERT. We learn self-attention consistent representations and extend BERT-like training to reinforcement learning using contrastive objectives.
- We explore the application of hierarchical reinforcement learning to train BERT, providing a direction for the combination of language models and reinforcement learning in the future.

2. Literature review

Deep reinforcement learning [38] combines the representational ability of deep learning with the reasoning ability of reinforcement learning. Its application to combinatorial optimization (CO) problems allows for processing larger-scale data with stronger robustness and generalization capabilities. Reinforcement learning is inherently designed to solve sequential optimization problems, and any CO problem can be equivalently transformed into a sequential optimization problem [38]. The main motivations for using deep reinforcement learning in CO are to approximate and discover new policies.

The output of traditional heuristics is an optimal action (decision variable). In contrast, the output of reinforcement learning is an optimal policy, which can effectively overcome various perturbations in the system compared to the optimal action [38]. In the case of perturbations, reinforcement learning often does not require re-optimization from scratch. However, heuristic search algorithms such as Bayesian optimization, particle swarm, and genetic algorithms must optimize the objective entirely from scratch. Because of the learned data distribution, reinforcement learning makes it easier to transfer optimization between similar optimization problems compared with traditional heuristic algorithms.

2.1. Related deep learning approaches

Applying neural networks to CO problems can be traced back to Hopfield & Tank [38], who use it to solve small-scale TSP problems. Several recent studies apply neural networks to CO. For example, Wu et al. [41] developed a deterministic annealing neural network (DANN) method based on the Lagrangian barrier function and two neural network models to solve the production transportation problem. Based on these two neural network models, they developed an iterative process and obtained search directions, then constructed two Lyapunov functions corresponding to the two neural network models. Wu et al. [42] proposed a DANN approach to obtain approximate solutions for graph partitioning. Their DANN algorithm is a continuation method that attempts to identify high-quality solutions by following a path of minimum points to the obstacle problem as the obstacle parameter decreases from a sufficiently large positive number to 0. We next list some deep learning techniques relevant to this paper as follows:

Pointer Networks. Vinyals et al. introduce a pointer network based on a sequence-to-sequence (seq2seq) network to learn the conditional probability of an output sequence. The elements are discrete labels corresponding to the positions in the input sequence [35]. It employs attention as a pointer to select an element of the input sequence as the output by obtaining the weights associated with the elements at a specific position of the output sequence and each position of the input sequence and then combining the input sequence and the weights in a certain way to produce the output.

Transformers. Most models use an encoder-decoder framework to deal with common seq2seq problems, mainly recurrent or convolutional neural networks. However, convolutional and recurrent neural networks suffer from low computing efficiency, insufficient parallel computing capacity, and overfitting, making them unsuitable for large-scale data. To solve these problems, Vaswani et al. propose a simple network structure called a transformer [34] framework, which does not require recurrent or convolutional layers but completely describes the global dependence of the input and output using an attention mechanism.

BERT. BERT [8] is a language model based on a multi-layer bidirectional transformer with two steps: pre-training and fine-tuning. First, BERT is pre-trained on two tasks, masked language modeling and next sentence prediction, using the BooksCorpus and English Wikipedia. Devlin et al. [8] then fine-tuned BERT on 11 natural language processing (NLP) tasks, achieving state-of-the-art performance on each.

Contrastive Learning. In certain circumstances, contrastive learning outperforms supervised learning as a recently developed unsupervised representation learning approach [28]. These techniques have also been used to boost performance in a reinforcement learning scenario. In addition to MRA [11], discussed above, CURL [22] combines Q-Learning with a different encoder for representation learning. Recently, contrastive learning has been used to define policy similarity embeddings, develop abstract representations of state-action pairings, and forecast future latent states. Various research has effectively coupled typical ResNet-50 architectures and contrastive estimates. SimCLR [4], in particular, relies on several powerful augmentations, whereas He et al. [16] use a memory bank. Inspired by target networks in reinforcement learning, Grill et al. [14] introduced BYOL: an algorithm for self-supervised learning that does not need a contrastive objective.

2.2. Reinforcement learning-based methods

Next, we outline previous works applying reinforcement learning to sequential decision optimization in CO problems.

Bello et al. [2] use an actor-critic architecture called neural combinatorial optimization (NCO) to solve the CO problem based on pointer networks. They consider two-stage methods based on policy gradients: the first stage is RL pre-training, which uses the training set to optimize the pointer network. The second stage is the active search, which starts with a random policy and optimizes the pointer network parameters on a single test instance with the expected return as the target while tracking the best solution sampled in the search. However, the actor-critic algorithm is notoriously unstable, and the RNN in the pointer network also limits the computational efficiency and generalization of the model.

Nazari et al. propose an end-to-end framework [29] that applies RL to solve vehicle routing problems based on pointer networks and NCO [2]. Their structure can handle any problem sampled from a given distribution rather than training a sep-

arate model for each problem instance. They take note of the dynamic nature of VRP and propose an alternative pointer network that can effectively deal with the system's static and dynamic elements. At each time step, the static element embedding serves as the input to the RNN decoder. The RNN output and dynamic element embedding are fed into the attention mechanism, forming a distribution on the optional feasible nodes. Although it can deal with dynamic sequence problems, it has similar problems to Bello et al.'s implementation [2]. The actor-critic sequence processing model based on RNNs limits generalization and computational efficiency.

Dai et al. [7] combined reinforcement learning and graph embeddings to solve CO problems on graphs. They use structure2vec to represent the policy in the greedy algorithm, which specializes in the nodes in the graph and captures the attributes in the context of the node's neighborhoods. This allows the policy to distinguish nodes based on their usefulness and generalize them to problem instances of different sizes. Still, previous sequence-based methods did not fully utilize the graph structure. They then used fitted Q-learning to learn a greedy policy parameterized by graph embedding. The overall effect directly optimizes the objective function of the original problem instance. They introduced a graph neural network (GNN) to learn the global information of the graph better to learn the internal structure of the combinatorial optimization problem. However, a problem with traditional GNNs is that when the scale of the problem instance increases, the number of parameters will increase significantly, making the model too heavy to be trained. Mittal et al. [27] trained a graph convolutional network (GCN) to solve large-scale problems, such as minimum vertex cover and the maximum cover problem, using the strategy of supervised learning and then reinforcement learning. They employ a greedy algorithm to train the neural network to embed the graph, predict the next node to choose at each stage, and then use a deep Q-network to train it further. However, their method relies on hand-designed heuristics to aid in training.

Kool et al. [20] applied a deterministic greedy rollout as a baseline (preferable to the value function in RL) to guide an improved transformer to learn the solution to the CO problem. Their policy network is also composed of an encoder and a decoder. They used the encoder to learn the representation of each node, the entire graph itself, and the decoder to predict the paths. The encoder utilized a graph attention network (GAT) [38], whose input is the coordinates of each node and whose output is the representation of each node. The representation of the graph is the average value of all node embeddings. The decoder also used a GAT, and input can be a node embedding, a context, a graph embedding, or a combination of the starting node embedding and the previous node embedding. The decoder's output is a series of nodes with the highest compatibility selected and added to the tour in each step. They masked visited nodes to ensure the feasibility of the solution. However, their method lacks flexibility because they can only generate solutions by decoding each node one at a time and cannot fine-tune and improve the trained policy. The quality of the solution is still slightly worse than state-of-the-art solution solvers. In contrast, we add self-supervised contrastive learning with hierarchical reinforcement learning to improve the training efficiency of the transformer. We introduce ReZero [1] into the transformer to optimize the policy network architecture and use BERT fine-tuning to introduce a contrastive learning loss to improve the solutions' quality further.

There are still other methods for CO that integrate deep learning (such as GNNs, RNNs, and attention) and RL [26,10,25,5,33,45,21,37,46,23,49,19,39]. Most have an encoder and decoder representing path-finding and path-reasoning. Learning-based methods can learn the data distribution of problem instances while simulating the heuristic algorithm process. Current learning-based methods are mainly based on construction heuristics [43,18,3] or improvement heuristics [40,6,25,48,44]. The former generates a sequence of solutions from scratch one by one, while the latter continuously improves the quality of the solution by improving a given initial solution. However, construction heuristic-based methods are still inferior to solution solvers in terms of quality. Meanwhile, improvement heuristic-based methods require a large number of iterations, and the quality of the final solution is heavily dependent on the given initial solution.

The most significant difference between our method and the above learning-based approaches is that we organically integrated reinforcement learning and self-supervised learning by combining BERT's pre-training and fine-tuning to bridge construction and improvement heuristics. To better utilize the masks in BERT, we introduce contrastive learning loss functions in the pre-training and fine-tuning phases to assist and enhance reinforcement learning and cross-entropy supervised learning, respectively. We can learn policies to generate solutions and fine-tune BERT to improve policies and enhance solutions. We effectively combine the advantages of construction heuristics that can quickly generate feasible solutions and improvement heuristics that continuously improve the quality of the solutions. This contributes to both the flexibility of learning policies and the improvement of solutions.

3. Methodology

3.1. Problem definition and preliminaries

An essential task of CO problems on graphs is path optimization. For example, the more complex facility location, vehicle scheduling, and vehicle-filled problems are based on the vehicle routing problem with additional optimization goals, decision variables, and constraints. Therefore, we take the capacitated vehicle routing problem (CVRP), the most basic and most studied variant of VRP with practical significance, as an example and describe the CO problem on the graph in detail.

Suppose there are k vehicles in a distribution center, the maximum load of each vehicle is Q, and n customers (nodes) need to receive goods. Each vehicle starts from the distribution center to deliver goods to several customers and returns to the distribution center. The demand for goods at customer point i is q_i $(i = 1, 2, ..., n)$, and $q_i < Q$, and we let point 0 rep-

resent the distribution center. d_{ij} represents the distance between customer i and customer j. How should we arrange vehicle routes to minimize the number of vehicles and the total distance traveled by the vehicles?

Let x_{ijk} and y_{ki} be decision variables. The following is a mathematical description of the problem:

$$x_{ijk} = \begin{cases} 1, & \textit{Vehicle } k \textit{ goes from } i \textit{ to } j \\ 0, & \textit{Otherwise} \end{cases}$$

$$y_{ki} = \begin{cases} 1, & \textit{Vehicle } k \textit{ accesses } i \\ 0, & \textit{Otherwise} \end{cases}$$

$$min \sum_i \sum_k \sum_j c = d_{ij} x_{ijk} \tag{1}$$

$$s.t. \sum_k y_{ki} = 1, \ (i = 0, 1, \cdots, n) \tag{2}$$

$$\sum_i x_{ijk} = y_{kj}, \ (j = 0, 1, \cdots, n; k = 1, 2, \cdots, m) \tag{3}$$

$$\sum_i x_{jik} = y_{kj}, \ (j = 0, 1, \cdots, n; k = 1, 2, \cdots, m) \tag{4}$$

$$\sum_i q_i y_{ki} \leq Q, \ (k = 1, 2, \cdots, m) \tag{5}$$

Equation (1) is the objective function, minimizing the total distance (cost) traveled. Equation (2) means that a vehicle only visits each node once. Equations (3) and (4) indicate that only one vehicle can reach and leave a specific node. Equation (5) is the capacity constraint of the vehicle.

Objective. Given a graph optimization problem G and a distribution D of problem instances, learn better heuristics to solve problems on an unseen graph G generated from D.

Motivation. Heuristics are often expressed as rules that can be interpreted as policies for making decisions. These policies can be parameterized by neural networks and trained to obtain new algorithms for many CO problems. This is a significant advance in that we can learn strong heuristics to solve many practical problems for which no good heuristics exist. The approaches of machine learning to solve CO problems mainly include two aspects: (1) Machine learning can accelerate the speed of traditional optimization algorithms. (2) Approximate and discover new policies, that is, learn some experience searching for optimal policies through machine learning to improve the effectiveness of current heuristic algorithms.

We build a parameterized model that can describe the probability distribution of the feasible solutions [20,2] and then optimize this model through reinforcement learning. We first construct the input: given a problem instance s, which includes n nodes, let the characteristics of the $i-th$ node be represented by $node_i$. For VRP, the instance can be expressed in coordinates: $s = ((x_i, y_i))_{i=\{1, 2, \cdots, n\}}$, and the nodes are fully connected.

We define the solution set $\pi = (\pi_1, \pi_2, \cdots, \pi_n)$ as the set of permutations of the nodes. Our goal is to learn a stochastic policy $p(\pi|s)$ for selecting the solution π to the problem instance s, assigning a higher probability to the optimal solution π^+ and lower probabilities of the less optimal solutions π^-. The policy $p(\pi|s)$ can be decomposed by the chain rule and parameterized by θ as:

$$p_\theta(\pi|s) = \prod_{i=1}^{n} p_\theta(\pi(i)|s, \ \pi(< i)) \tag{6}$$

We regard an instance s as a graph and directly use the graph as the input, where coordinates characterize the information of the nodes. A GAT can process the structure of the graph through the masking process. Using the graph as input rather than a sequence of nodes is better because it eliminates the dependence on the order of the nodes given in the input. The learned heuristic does not depend on the orientation of the input instance s. In other words, no matter how we arrange the nodes, if the given graph is unchanged, the output will not change, which is an advantage over the sequence method.

VRP as a Markov decision process. We define path optimization as a Markov decision process (MDP) to apply reinforcement learning to path optimization problems. We define the tuple (S, A, T, R) of the MDP (state, action, transition, and reward) as follows:

State. The current state $s'_t \in S$ of the agent refers to the set of all possible actions (nodes) on the graph G, where t is the time step. Since we represent actions or nodes using embeddings, the states are p-dimensional vectors, and the final state will depend on the problem to be solved.

Action. The action space A is the selection set of the agent, while an individual action a_t is a node of graph G that does not belong to the current state S, which can be expressed as its corresponding p-dimensional node embedding. $A = (a_1, a_2, \cdots, a_n)$.

Transition. The transition is a function that updates the states according to the actions ($S \times A \rightarrow S$). Since the graph G is known and static, the MDP transition is known and determined. We define the policy using the transition probabilities of states and actions: $p_\theta(\pi|s) = p_\theta(a_t|s'_t)$.

Reward. From $p_\theta(\pi|s)$, the solution $\pi|s$ can be obtained. We use the solution length as the reward $R(\pi|s) = \sum_{t=1}^{n} R(s'_t, a_t) = -L(\pi|s)$, and our goal is to maximize $R(\pi|s)$, or minimize the loss $L_1 = L(\pi|s)$.

3.2. Policy network architecture

In the framework of BDRL, we employ a deep reinforcement learning algorithm to parameterize and train BERT. Then we can fine-tune the pre-trained model to apply it to other combinatorial optimization problems with different variables but similar structures.

Encoder. We first map the input (a token sequence) to a low-dimensional embedding space and then input it to the transformer's neural network. The output is a sequence of vectors of size H, and each vector corresponds to a token with the same index. The output of the encoder in BDRL (Fig. 1) is a set of action vectors $A = (a_1, a_2, \cdots, a_n)$, each of which represents an input node ($node_i$) interacting with other nodes in the context. As mentioned above, the transformers in BERT are bidirectional. This encoder has a more robust encoding capability than the encoder of the original single transformer because it can process the bidirectional context information on each node, similarly to a GAT. Specifically, it allows the input sequence to be treated as a graph without the need to improve the transformer's encoder to make it more GAT-like, as Kool et al. [20] have done. The BDRL encoder's information includes (1) encoder embeddings and (2) context vectors.

ReZero transformer encoder layer. The transformer encoder in BDRL is similar to that in [20], which consists of N identical layers, each with two sub-layers, one for multi-head attention (MHA) and one fully connected feedforward (FF) layer. We apply the state-of-the-art ReZero [1] to the transformer (as shown in Fig. 2, to the left is an overview of one of the layers of the ReZero transformer and to the right is the processing of node embeddings in the MHA and FF layers) to solve the problem of poor signal propagation caused by multi-head attention, which also allows the model to learn deeper networks with faster convergence than batch normalization and layer normalization. Embeddings in the encoder include node embeddings and graph embeddings, and the embedding process in the two sub-layers and message passing between nodes is as follows:

$$\widehat{e}_i = e_i^l + \alpha_i^l \cdot MHA_i^{l+1}\left(e_1^l, \cdots, e_n^l\right) \tag{7}$$

$$e_i^{l+1} = \widehat{e}_i + \alpha_i^l \cdot FF^l\left(\widehat{e}_i\right) \tag{8}$$

Multi-head attention. The transformer [34] employs an attention mechanism to build feature vectors for each node and determine how important the other nodes in the graph are to that node. The update of a node feature is the sum of the linear transformations of the other node features, weighted according to their importance. For example, let s represent the node set in the instance, while Q, K, and V are learnable linear weights and represent the query, key, and value of the attention calculation. We update the hidden feature e of $node_i$ from layer l to layer $l+1$ in the graph s as follows:

$$e_i^{l+1} = Attention\left(Q^l e_i^l, K^l e_j^l, V^l e_j^l\right) = \sum_{j \in S} softmax_j\left(Q^l e_i^l \cdot K^l e_j^l / \sqrt{d}\right) \cdot \left(V^l e_j^l\right) \tag{9}$$

In a transformer, the attention mechanism is executed in parallel for each node in the instance, while node updates in an RNN model are performed node by node [35].

Getting multi-head attention to work is difficult because random initialization can destabilize the learning process. We can overcome this problem by executing the attention of multiple "heads" in parallel and linking the results so that each

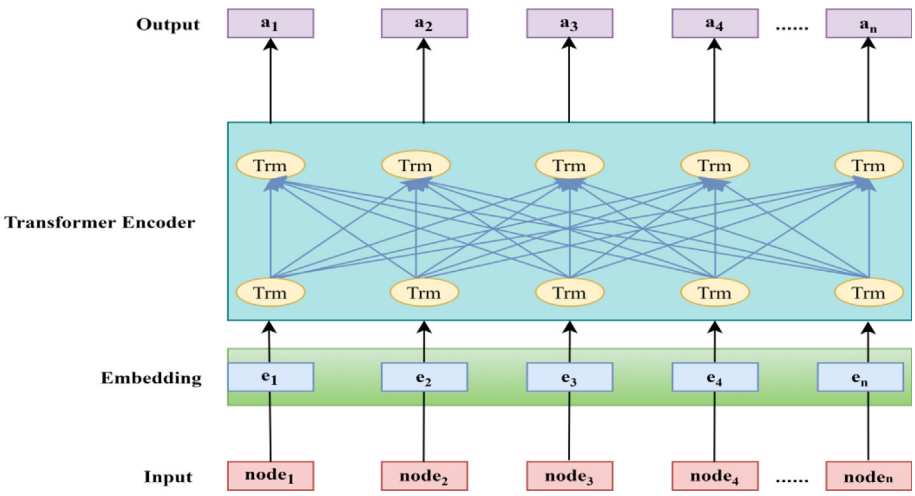

Fig. 1. The encoder of BDRL, in which "Trm" represents the transformer.

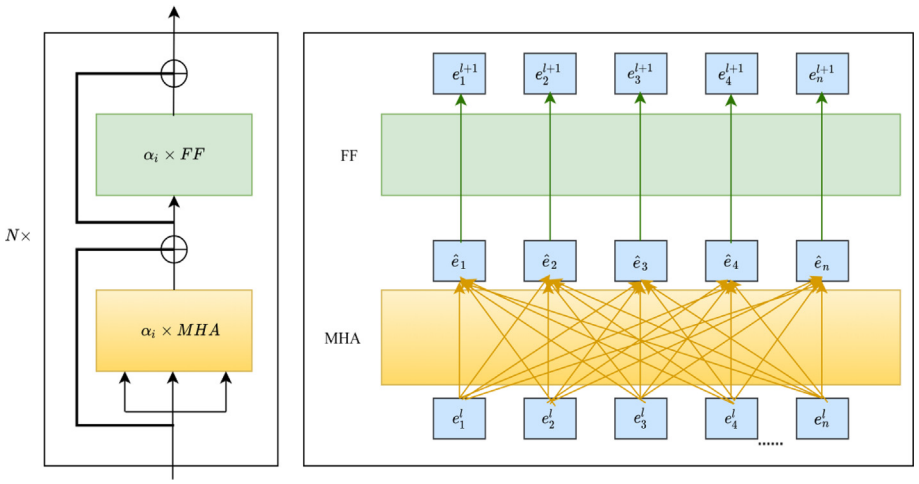

Fig. 2. The encoder layer of the transformer in BDRL.

head has a variable weight. Let $Q = [q_1, q_2, \cdots, q_n]$, $K = [k_1, k_2, \cdots, k_n]$ and $V = [v_1, v_2, \cdots, v_n]$ be the learnable weights of the $k-th$ attention head, and O^l be the downward projection to match the cross-layer dimensions of e_i^{l+1} and e_i^l. Then:

$$e_i^{l+1} = Concat(head_1, \cdots, head_K)O^l \tag{10}$$

$$head_k = Attention\left(Q^{k,l}e_i^l, K^{k,l}e_j^l, V^{k,l}e_j^l\right) \tag{11}$$

The pointing mechanism. BERT outputs a contextualized vector representation for each input word, and we can get pooled output or sequence output depending on the specific NLP task [8]. In BDRL, we choose to get sequence output to predict the probability distribution of the nodes. We use the same pointing mechanism as in [20,2] to compute the node distribution ($p_\theta(\pi(i)|s, \pi(<i))$) of the encoded nodes (actions) and query vectors (state representations); that is, we pass the pointing vector u_j output to the softmax layer to generate the distribution of the next candidate node. Specifically:

$$u_j^i = \begin{cases} v^T \cdot \tanh\left(W_{ref}a_i + W_q q_i\right), & i \notin \{\pi_0, \pi_1, \cdots, \pi_{i-1}\} \\ -\infty, & otherwise \end{cases} \tag{12}$$

$$p_\theta(\pi(i)|\pi(<i), s) = softmax(Ctanh(u^i/T)) \tag{13}$$

Two attention matrices, W_{ref} and W_q, as well as an attention vector v, an action vector a_i, and a query vector q_i, parameterize the pointing mechanism in equation (12). In equation (13), we also use the control of entropy method [2] in the clipping logic of $[-C, +C]$, where T is a temperature hyperparameter used to control the certainty of sampling, such that $T = 1$ when the model is being trained, and $T < 1$ during inference.

Contrastive learning for RL pre-training. RL is more suitable for combinatorial optimization problems than supervised learning because the large amount of labeled data required is expensive and not readily available [2,7]. Furthermore, RL's universality and generalization capabilities are more robust than supervised learning. Evidence shows that learning from low-dimensional state-based features is substantially more effective. Empirically, performing RL directly from high-dimensional observations is sample inefficient. It could be feasible to learn from raw observations as quickly as from states if state information could be successfully retrieved. However, unlike in the supervised or unsupervised context, non-stationarity in the training distribution and high correlation between observations at adjacent timesteps make learning representations in RL more challenging.

Furthermore, learning representations in RL must be accomplished with little to no supervision because of the frequently sparse reward signal from the environment. The methods used to deal with these problems may be divided into two classes. The first class leverages supplemental self-supervised losses to quicken model-free RL algorithms' learning efficiency [22,18]. The second class studies a world model and employs it to gather fictitious rollouts, which serve as additional data to train the RL algorithm and minimize the number of environmental samples needed. BDRL is in the first class since it employs a self-supervised loss to boost data effectiveness. We are primarily motivated by the impressive developments in contrastive learning and BERT [8] during the past few years. We use the bidirectional processing of transformers with a masked prediction setup from BERT. With this pairing, the model must concentrate on the context of nearby timesteps to achieve its objective [36]. Therefore, the model attends to all the pertinent frames in the trajectory when requested to reconstruct a specific

masked frame. Paying attention to other frames in the sequence may give more evidence to lessen this uncertainty. This is especially relevant in RL since state aliasing is a source of uncertainty in value estimates.

But in contrast to BERT, where objectives are provided, and the input is a discrete vocabulary for language learning, in RL, the inputs are rewards, states, and actions that do not constitute a discrete or limited collection. As a result, we must create proxy targets and the related proxy tasks to complete. To do this, we employ contrastive learning, and our contrastive loss is derived from RELIC [28]. To increase generalization guarantees, RELIC uses the original data to generate several augmentations, which are subsequently used to enforce invariant prediction of proxy targets across augmentations. However, unlike RELIC, BDRL does not make use of data augmentations. Instead, we rely on the input data's sequential character to arrange comparable and dissimilar points into the proper categories for contrastive learning. For RL, the absence of augmentations is essential since creating customized augmentations for each domain would be labor-intensive.

Auxiliary loss. Before feeding embeddings into the transformer stack, 15 % of the embeddings in a batch of sampled sequences are replaced with a fixed token designating masking. Then, let $m \in \Gamma$ and the set Γ to represent the sequence's randomly masked indexes. Let a_m^i be the transformer encoder's output for the $i-th$ training sequence in the batch, and let e_m^i be the matching input to the transformer encoder for each index $m \in \Gamma$. Let $\varphi(a,e) = \psi(a)^T\psi(e)$, where ψ is a critic function (a single layer MLP of size 512), be the inner product identified on the space of critic embeddings. The critic distinguishes between the embeddings used for the downstream RL problem and the contrastive proxy task. Because the proxy and downstream tasks are connected but distinct, it is necessary to separate them because otherwise, the appropriate representations might not be the same. The critic can be used to decrease the dimensionality of the dot-product as a side effect. We use e_m^i as a positive example, and the sets $\{e_m^b\}_{b=0,b\neq i}^B$ and $\{a_m^b\}_{b=0,b\neq i}^B$ as the negative examples, where B represents the number of sequences in the minibatch, to learn the embedding a_m^i at mask points m. We define x_e^m (x_a^m is computed analogously) as:

$$x_e^m = \frac{exp(\varphi(a_m^i, e_m^i))}{\sum_b^B exp(\varphi(a_m^i, e_m^b)) + exp(\varphi(a_m^i, a_m^b))} \tag{14}$$

We formally regularize the similarity between the pairs of transformer embeddings and inputs using Kullback-Leibler regularization from RELIC to impose self-attention consistency in the learned representations. We examine the similarity within each set of acceptable embeddings and inputs and between the two. In light of this, we define:

$$\mu_a^m = \frac{\exp\left(\varphi(a_m^i, e_m^i)\right)}{\sum_b^B \exp\left(\varphi(a_m^i, e_m^b)\right)}, \quad \xi_a^m = \frac{exp(\varphi(a_m^i, a_m^i))}{\sum_b^B exp(\varphi(a_m^i, a_m^b))} \tag{15}$$

μ_e^m and ξ_e^m are defined analogously. The overall loss is expressed as follows, where sg denotes the stop-gradient:

$$\begin{aligned} L(A,E) = &-\sum_{m\in\Gamma}(logx_a^m + logx_e^m) \\ &+ \alpha\sum_{m\in\Gamma}\left[KL(\xi_a^m, sg(\xi_e^m)) + KL(\mu_a^m, sg(\mu_e^m)) + KL(\mu_a^m, sg(\xi_e^m)) + KL(\mu_e^m, sg(\xi_a^m))\right] \end{aligned} \tag{16}$$

Similarly to our RL objective, we optimize this contrastive objective using the whole batch of sequences collected from the replay buffer. Finally, we optimize the sum of $L(A,E)$ and RL objective $J(\theta)$:

$$L_1' = L(A,E) + J(\theta) \tag{17}$$

Hierarchical RL for VRP. Inspired by [26] and considering the complexity of BERT, we employ hierarchical RL [17] to pretrain BDRL hierarchically and deal more effectively with constrained VRP problems. The core idea of hierarchical RL [17] is to combine some primitive action sequences into a complex macro action. Most algorithms facilitate the abstraction of this behavior by establishing a "manager-learner" framework, which is usually a collection of top-level policies. These collections are trained to select from a set of policies that have been trained to achieve certain sub-goals. Hierarchical RL can significantly reduce the action space size and training iteration time and improve generalization ability.

We use a similar hierarchical RL definition and algorithm as [26] in our model, sampling actions from a policy defined at each level of the hierarchy. For a given layer $k \in \{0,\cdots,K\}$, we extract the solution $\pi^{(k)}$ from the policy $p_{\theta_k}\left(\pi_{\widehat{i}}^{(k)} | s_{\widehat{i}}^{(k)}, h_{\widehat{i}}^{(k)}\right)$, where $h_{\widehat{i}}^{(k)} \in H^{(k)}$ represents the latent variable of the $k-th$ layer, and $H^{(k)}$ is its corresponding latent space. For the hierarchical policy, the objective function of the $k-th$ layer is:

$$J(\theta_k) = -E_{\pi\ p_{\theta_k}(\pi)}\left[L\left(\pi^{(k)}|s^{(k)}\right)\right] \tag{18}$$

We employ a policy gradient method based on REINFORCE [38] to optimize the hierarchical policy and use a baseline similar to [20], where a solution $\widehat{\pi}_{\widehat{i}}^{(k)}$ is sampled from the greedy policy $p_{\theta_k}^{Greedy}$:

$$b_{\widehat{i},k}(s) = \ L_k\left(\widehat{\pi}_{\widehat{i}}^{(k)} | s_{\widehat{i}}^{(k)}\right) + \frac{1}{B}\sum_{j=1}^{B}\left(L_k\left(\pi_j^{(k)} | s_j^{(k)}\right) - L_k\left(\widehat{\pi}_{\widehat{i}}^{(k)} | s_{\widehat{i}}^{(k)}\right)\right) \tag{19}$$

We summarize the complete flow of the layer-wise policy optimization in Algorithm 1.

Search. As mentioned above, the general idea of solving path optimization problems on graphs is to find the paths and then reason them. More specifically, we obtain a set of feasible solutions through path-finding and then the optimal solution through path reasoning. It is not expensive to employ RL to evaluate the length of a feasible path. The agent can easily simulate a search process in the path reasoning stage to search for the best one from a large set of feasible solutions. As mentioned in [2], when comparing different search strategies, the effect of sampling is more stable than greedy and active search, so we use sampling as the BDRL search strategy.

Algorithm 1 Policy Optimization

Input: a policy network $p_{\theta_k}(\pi|s)$, training steps $N_0, N_1, \ldots, N_T$, training set ξ, learning rate α, batch size B, and number of layers K

1: Initialize policy network parameters $\theta_k (k \in \{0, \cdots, K\})$
2: for $m \in [1, N_T]$ do
3: for $k \in [0, K]$ do
4: $s_1, s_2, \cdots, s_B$ $SampleInput(\xi)$
5: for $j = 0$ to $k-1$ do
6: $\pi_{\widehat{i}}^{(j)}, h_{\widehat{i}}^{(j+1)}$ $SampleSolution\left(p_{\theta_k}\left(.|s_{\widehat{i}}^{(j)}, h_{\widehat{i}}^{(j)}\right)\right)$
7: $\pi_{\widehat{i}}^{(k)}$ $p_{\theta_k}\left(.|s_{\widehat{i}}^{(k)}, h_{\widehat{i}}^{(k)}\right)$
8: $\widehat{\pi}_{\widehat{i}}^{(k)}$ $p_{\theta_k}^{Greedy}\left(.|\widehat{s}_{\widehat{i}}^{(k)}, h_{\widehat{i}}^{(k)}\right)$
9: Update policy network parameters:

$$\nabla_{\theta_k} J(\theta_k) \leftarrow \frac{1}{B}\sum_{\widehat{i}=1}^{B}\left[\left(L_k\left(\pi_{\widehat{i}}^{(k)}|s_{\widehat{i}}^{(k)}\right) - b_{\theta_k}\left(s_{\widehat{i}}^{(k)}\right)\right)\nabla_{\theta_k} logp_{\theta_k}\left(\pi_{\widehat{i}}^{(k)}|s_{\widehat{i}}^{(k)}, h_{\widehat{i}}^{(k)}\right)\right]$$

10: $\theta_k \leftarrow Adam(\theta_k, \nabla_{\theta_k} J(\theta_k))$
11: Return $\theta_1, \theta_2, \cdots, \theta_K$

3.3. Fine-tuning

The fine-tuning of BERT [8] typically adds a task-specific output layer and then trains the entire BERT model based on labeled data. To get the most out of the contextual representations with rich node information, we fine-tune the pre-trained BDRL for combinatorial optimization. It is worth noting that we can already use the pre-trained BDRL for reasoning in a problem-specific test set. At the same time, fine-tuning can lead to better model generalization because it can be transferred to other problems using only some labeled data.

We apply the solutions to the combinatorial optimization problem (such as the feasible paths in VRP) as the input "sentence". We represent a path as a sequence of nodes, similar to how a sequence of tokens is input to BERT. We illustrate the BDRL fine-tuning architecture of modeling path sequences in Fig. 3.

The first token of each input in the fine-tuning model is a unique classification token [CLS]. We fine-tune the model with solutions as input, where each solution π is represented by a sequence of paths $(\pi_1, \pi_2, \cdots, \pi_n)$ that contain tokens such as "012340", "021430", or "031240", and each path is separated by a separator token [SEP]. We build the representation of a given sequence of input tokens by adding the corresponding tokens' embedding, segments, and positions. Different path sequences divided by [SEP] have different segment embeddings, and different tokens at the same location have the same location embedding. We use the hidden vector of the final output corresponding to the [CLS] token, $C \in R^H$, as the aggregate sequence representation, while the vector corresponding to the $\bar{i}-th$ input token is denoted as $O_{\bar{i}} \in R^H$, where H is pre-trained hidden state size in BERT.

The scoring function (conditional probability) of the solutions is $z_\pi = f(\pi_1, \pi_2, \cdots, \pi_n) = sigmoid\left(CW^T\right)$, which is a 2-dimensional vector with $z_{\pi_0}, z_{\pi_1} \in$ and $z_{\pi_0} + z_{\pi_1} = 1$, where $W \in R^H$ is a learnable classification layer weight vector. We use z_π and the solution labels to calculate the cross-entropy (CE) loss as follows, where $y_\pi \in \{0, 1\}$ is the solution label, π^+ is the set of "good" solutions, and π^- is the set of "bad" solutions:

$$L_{CE} = -\sum\nolimits_{\pi \in \pi^+ \cup \pi^-} \left(y_\pi \log\left(z_{\pi_0}\right) + (1 - y_\pi)\log\left(z_{\pi_1}\right)\right) \tag{20}$$

The purpose of fine-tuning is to categorize the solutions for the instance s as being "good" or "bad". For VRP, we can directly obtain the policy network through pre-training, which generates multiple tours in parallel in the fine-tuning stage.

Contrastive learning aims to bring similar data points closer and unrelated ones farther away. We introduce a novel contrastive learning loss L_{SCL}, which treats samples of the same class as positive examples to each other and different classes as

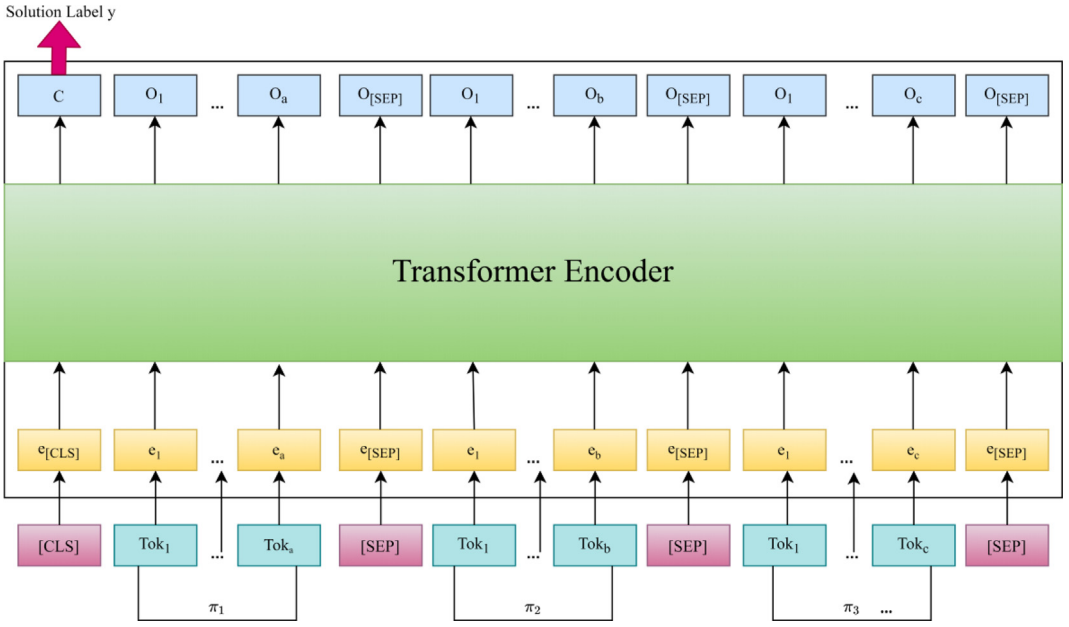

Fig. 3. Illustration of fine-tuning BDRL for solution classification.

negative examples to achieve the purpose of bringing the intra-class samples closer and distancing the inter-classes, specifically:

$$L_{SCL} = -log \frac{e^{sim\left(\pi_i, \pi_i^+\right)/\tau}}{\sum_{j=1}^{n}\left(e^{sim\left(\pi_i, \pi_j^+\right)/\tau} + e^{sim\left(\pi_i, \pi_j^-\right)/\tau}\right)} \tag{21}$$

Here, τ controls the temperature, e stands for exponent, and *sim* calculates the cosine similarity. The total loss in the fine-tuning phase is the sum of the cross-entropy and contrastive learning loss:

$$L_2 = L_{CE} + L_{SCL} \tag{22}$$

Since the MDP is not involved in our fine-tuning stage, L_2 cannot be combined with the L_1' for policy gradient optimization. Instead, we use Stochastic Gradient Descent (SGD) to update W and the pre-trained parameter weight ∇L_2. In the fine-tuning phase, Bello et al. [2] apply reinforcement learning to fine-tune policy network parameters using reward and punishment information to obtain the best tour path. However, BDRL can be fine-tuned using only specific data (solutions) in a supervised or self-supervised manner.

Additionally, the two parts of BDRL, pre-training and fine-tuning, can be completely independent. As mentioned above, we only need the pre-trained model to reason with specific problems. On the other hand, we can also fine-tune the pre-trained model to adapt it to the decision-making classification problem. We can also add a search strategy to search for feasible candidate paths and only need supervised learning to train it.

4. Experiments

4.1. Implementation and setup

We conduct the experiments on eight NVIDIA GeForce GTX1080Ti GPUs. In the pre-training phase of algorithm 1, we set the maximum number of epochs at 100 (with early stopping if the shortest path obtained stabilizes), epoch size at 1280000, batch size at 512, evaluation batch size at 1024, and training steps at 2500. We employ the Adam optimizer to optimize parameters and set the learning rate and learning rate delay at 10^{-3} and 0.96, respectively. We set the contrastive loss weight to 1 and the contrastive loss mask rate to 0.15 in pre-training and fine-tuning. In the encoder, we set the initial α_i of the ReZero transformer to 0, the number of layers N of the encoder layer of the transformer to 3, and the input and output dimensions of the feedforward sub-layer to $d = 128$. Each layer has $h = 6$ parallel heads and $d = 128$ hidden dimensions, and the dimension of its inner layer is $4d = 512$. We use the same setting for all tasks in the experiment.

We focus our experiment on two problems: TSP and VRP, as well as their variants CVRP, the orienteering problem (OP), and the prize collection TSP (PCTSP). We train BDRL using 1000 Euclidean TSP20, TSP50, TSP100, VRP20, VRP50, and VRP100 graph benchmark sets [20]. Regarding the encoder of BDRL as a kind of GNN, we also retrain the model on small graphs and then reason on larger graphs, like Dai et al. [7].

It is known that the number of parameters in the BERT framework is vast, which generally consumes a lot of computing resources. To address this problem, Google researchers designed ALBERT (A Lite BERT) [13], which has far fewer parameters than the traditional BERT architecture. ALBERT overcomes the main obstacles by expanding the pre-trained model through two parameter-reduction techniques. One is to factorize embedding parameterization, and the other is cross-layer parameter sharing. The configuration of ALBERT is similar to BERT-large, but the number of parameters is only 1/18 of the latter, and the training speed is 1.7 times that of the latter. These parameter reduction techniques can also act as some form of regularization, making training more stable and facilitating generalization. Therefore, to further reduce the burden of model training and prevent overfitting, we use ALBERT as the basic framework in our experiments.

4.2. Performance on small-scale TSP

We train BDRL with TSP20, TSP50, and TSP100 instances [20]. We use the optimal (shortest) path length (Obj.), gap (the best value across all methods), and inference (test) time to evaluate model performance. We compare the performance of BDRL on a small TSP with three types of baselines: solution solvers such as Concorde, Gurobi, LKH3, and OR Tools; heuristic methods such as nearest insertion, random insertion, farthest insertion, and MST [20]; and RL-based methods relevant to our work, such as GPN [26], attention model (AM) [20], S2V-DQN [7], and PN-AC [2]. Table 1 shows the comparison results, and we take some baseline experimental results from [20,7].

Fig. 4 shows that BDRL has the lowest approximation ratio to the optimal solution compared to traditional heuristic algorithms and recent learning-based methods. Learning-based methods are more accurate than traditional heuristic algorithms like nearest insertion, MST, and other insertion algorithms. However, Table 1 shows that the current state-of-the-art solution solver has obtained the optimal solution on a small-scale TSP instance. The learning-based methods in the baseline are all based on construction heuristics, while solution solvers like Concorde and LKH3 continuously improve the quality of the initial solution. Therefore, it is reasonable that algorithms based on learning heuristics are inferior to solution solvers in terms of solution quality. Still, once we train the model, it takes a much shorter time than the solution solvers to test. More importantly, the learning method has much better generalization ability, and the solution solvers can only solve specific static combinatorial optimization problems in a targeted manner. BDRL is superior to other learning-based methods in small-scale TSP instances in terms of accuracy and time. We believe this can be attributed to modeling, training, and the fusion of the two, which we will further verify in subsequent experiments.

4.3. Resolving VRP and other problems

For solving path optimization problems with constraints (such as CVRP), we follow [20] to generate VRP instances for $n = 20/50/100$ and normalize the demands of each node according to their capacities. Each node in the graph is randomly sampled from $(0 \times 1]^2$, where $(0, 0)$ represents the warehouse or central node. We define the demand for each delivery point as $q_i' = \frac{q_i}{Q^n}$, where q_i is uniformly sampled from $\{1, \cdots, 9\}$ and Q is vehicle capacity, where $Q^{20} = 30$, $Q^{50} = 40$, and $Q^{100} = 50$.

To satisfy the capacity constraints as in [35], we track the remaining demand $q_{i,t}'$ of node i and the remaining vehicle capacity Q_t' at each time step t. At the beginning $(t = 1)$, $q_{i,t}'$ is initialized as q_i', and $Q_t' = 1$, and then they are updated with the following formulas:

Table 1
Comparison of BDRL and baselines according to the shortest path, gap, and reasoning time. Unmeasured values are replaced by "-". Concorde and LKH3 solution solvers obtain the optimal solution, which serves as a benchmark for other models.

Method	TSP20			TSP50			TSP100		
	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time
Concorde	**3.84**	0.00 %	1 *m*	**5.70**	**0.00 %**	2 *m*	**7.76**	**0.00 %**	3 *m*
LKH3	**3.84**	0.00 %	18 s	**5.70**	**0.00 %**	5 *m*	**7.76**	**0.00 %**	21 *m*
OR Tools	3.85	0.37 %	–	5.80	1.83 %	–	7.99	2.90 %	–
Nearest Insertion	3.84	0.00 %	18 s	6.78	19.03 %	2 s	9.46	21.82 %	6 s
Random Insertion	3.85	0.37 %	–	6.13	7.65 %	1 s	8.52	9.69 %	3 s
Farthest Insertion	3.93	2.36 %	1 s	6.01	5.53 %	2 s	8.35	7.59 %	7 s
PN-AC	3.89	1.42 %	–	5.95	4.46 %	–	8.30	6.90 %	–
S2V-DQN	3.89	1.42 %	–	5.99	5.16 %	–	8.31	7.03 %	–
AM	3.85	0.34 %	**0 s**	5.80	1.76 %	2 s	8.12	4.53 %	6 s
BDRL	**3.84**	0.00 %	**0 s**	5.73	0.53 %	**1 s**	7.81	0.64 %	**3 s**

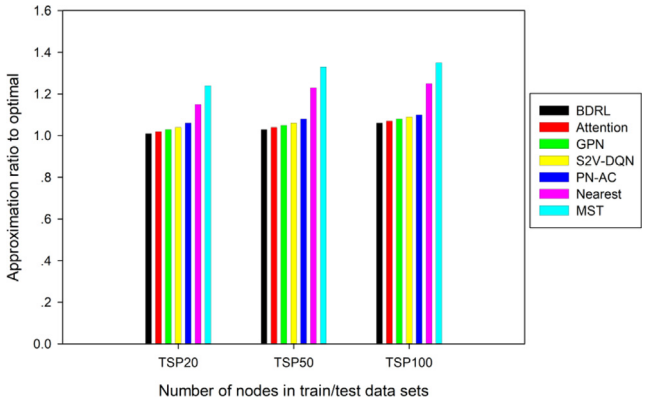

Fig. 4. The approximation ratio of the models to the optimal solution on TSP20, TSP50, and TSP100.

$$q'_{i,t+1} = \begin{cases} \max\left(0, q'_{i,t} - Q'_t\right), & \pi_t = i \\ q'_{i,t}, & \pi_t \neq i \end{cases} \tag{23}$$

$$Q'_{t+1} = \begin{cases} \max\left(Q'_t - q'_{\pi_t,t}, 0\right), & \pi_t \neq 0 \\ 1, & \pi_t = 0 \end{cases} \tag{24}$$

The orienteering problem (OP) [12] is a variant of TSP, which simulates many practical problems in the real world and is employed in banking, computer science, education, finance, government, healthcare, and other industries. Each node in the OP has a specific prize, and we aim to generate a route that traverses all nodes (from the warehouse and back) and maximizes the prize, but that is shorter than the given maximum length. OP is much more complex than TSP because the former has to maximize the prize while simultaneously ensuring that the route length is shorter than the threshold. We distribute the prizes such that greater prizes are found farther away from the origin.

In the prize collection TSP (PCTSP) [24], each node has prizes and penalties. Our goal is to collect at least a minimum reward while minimizing the penalty of not visiting nodes and the total tour length. Besides accounting for the reward and tour length, as in OP, our method must also consider the costs of visiting and not visiting nodes.

We evaluate the model performance on CVRP, OP, and PCTSP, with the number of nodes$n$ = 20/50/100, and demonstrate the results in Table 2. As we can use the pointing mechanism and different data to fine-tune BDRL for different problems, we also evaluate the performance of BDRL without fine-tuning and BDRL with only fine-tuning to show that fine-tuning does have an enhanced effect on the model (demonstrated in Table 3).

From the experimental results in Table 2, we can see that BDRL performs better than AM on the test sets of three problems with 20, 50, and 100 nodes. We only selected the best version of AM, the sampled version, instead of the greedy version, for a fair comparison. Compared with AM, which also relies on attention, the attention structure in BDRL is more complex, which may improve its capabilities. Hierarchical reinforcement learning has certain advantages for processing complex models with multi-level structures. We believe that modeling and training are essential factors that improve performance. The solution solvers still show good accuracy on specific problems, but their calculation times increase exponentially as the problem size increases. More importantly, it is not fair to use solution solvers as a baseline for learning-based methods because,

Table 2
We demonstrate the performance of different methods on different problem instances, in which we choose the best version of the baseline results.

Method	n = 20	n = 50	n = 100	Problems
	Obj. Time	Obj. Time	Obj. Time	
LKH3	**6.14** 2 h	10.38 7 h	**15.65** 13 h	
OR Tools	6.43 -	**11.31 -**	17.16 -	CVRP
AM	6.25 6 *m*	10.62 28 *m*	16.23 2 h	
BDRL	**6.14 4** ***m***	**10.38 12** ***m***	15.73 **1.5 h**	
Gurobi	13.026 33 s	**- -**	**- -**	
OR Tools	**4.09** 52 *m*	**- -**	**- -**	OP
AM	5.30 4 *m*	16.07 16 *m*	32.68 53 *m*	
BDRL	5.19 **3** ***m***	**15.67 9** ***m***	**31.74 46** ***m***	
OR Tools	3.26 0 s	**4.48** 5 h	**5.98** 5 h	
AM	3.26 0 s	4.65 2 s	6.32 5 s	PCTSP
BDRL	**3.26 0 s**	**4.48 1 s**	6.02 **3 s**	

Table 3
The effect of fine-tuning on BDRL.

Method	n = 20	n = 50	n = 100
	Obj. Time	Obj Time	Obj Time
BDRL	**6.14 4 *m***	**10.38 12 *m***	**15.73 1.5 h**
BDRL (w/o fine-tuning)	6.19 **4 *m***	10.56 14 *m*	16.09 1.6 h
BDRL (only fine-tuning)	6.30 6 *m*	10.67 17 *m*	17.48 2 h

for different problems, solution solvers need to call different algorithms to solve the problems in a targeted manner. In other words, solution solvers improve accuracy compared with learning-based methods, but at the expense of generalization.

Nevertheless, BDRL is very competitive in terms of accuracy and calculation time, and we can see that its advantages become more evident as the scale of the problems increases. We can also see that BDRL has good generalization capabilities. It can effectively learn the problems' internal structure and data distribution and extend flexibly to similar problems.

Table 3 shows the results of the BDRL fine-tuning ablation study on CVRP with 20, 50, and 100 nodes. We split BDRL into three forms: (1) the full version (all the modeling and training in the methodology section above). (2) w/o fine-tuning (use hierarchical reinforcement learning to train the policy network and decode immediately). (3) Only fine-tuning (train the original BERT in a supervised manner using only the labeled data). We can see that BDRL, with fine-tuning, achieves the best results.

We showed that we could train the model in a supervised manner based on introducing relevant labeled data in the fine-tuning stage to strengthen generalization capabilities. The good generalization ability for different problems in Table 2 proves the effectiveness of fine-tuning. We further verify the essential role of fine-tuning in BDRL through ablation experiments, showing that fine-tuning significantly improves the pre-trained model. Furthermore, even without fine-tuning, the results of BDRL are still superior to AM, but a model with only fine-tuning is far inferior to the other two cases of BDRL.

Table 4 shows that our contrastive loss contributes to significantly improved performance on CVRP20, CVRP50, and CVRP100 instances compared to BDRL without contrastive loss. Also, contrastive loss plays a more significant role in the pre-training stage than in the fine-tuning stage. As the scale of the data increases, the effect of contrastive learning becomes more apparent, proving that it plays a positive role in data sampling. Furthermore, in a challenging environment like CVRP, BDRL outperforms other learning-based baselines without contrastive loss. This may benefit from improvements to the transformer policy network and the efficiency of hierarchical reinforcement learning.

4.4. The effect of learning

To show the learning effect of BDRL more intuitively, we describe in detail the learning curve of BDRL in the training and testing process. We first train BDRL for 50 epochs on the TSP20 instance and get the average cost (shortest route length) after each epoch.

Fig. 5 (the left figure shows the curve without a contrastive loss (CL), and the right figure shows the curve with CL) shows the learning curves of BDRL on the TSP20 instance, from which we can see that as the epoch increases, the average route length obtained by BDRL gradually converges and stabilizes. The learning curve in the left figure is not very smooth, which is reasonable because reinforcement learning training is notoriously unstable. However, we can see that the learning curve with the CL has an apparent gradient descent trend and is more stable; even on a small-scale instance such as TSP20, contrastive learning has a noticeable auxiliary effect on the learning effect of reinforcement learning.

To observe how quickly our model learns, we graphed the results BDRL obtained at each time step (each time step contains 50 training steps, and each epoch contains 50-time steps (2500 training steps)) in the first eight epochs. Fig. 6 shows that the learning curve converges rapidly in the first epoch, while the learning curve in subsequent epochs converges more gradually toward the optimal solution.

The generalization of learning-based methods stems from the assumption that problem instances of the same type have similar data distributions. Therefore, we also experimented with applying a TSP100 pre-trained model to a different TSP100 data set to verify whether BDRL can effectively learn the data distribution of the new set. The running speed on TSP100 is significantly lower than on TSP20 because the combination space increases geometrically with an increase in the number of

Table 4
The impact of contrastive loss (CL) on BDRL.

Method	n = 20	n = 50	n = 100
	Obj. Time	Obj. Time	Obj. Time
BDRL (w/o CL)	6.18 8 *m*	10.53 18 *m*	16.11 2 h
BDRL (w/o CL in pre-training)	6.16 5 *m*	10.47 14 *m*	15.89 1.6 h
BDRL (w/o CL in fine-tuning)	6.17 7 *m*	10.50 16 *m*	15.95 1.7 h
BDRL	**6.14 4 *m***	**10.38 12 *m***	**15.73 1.5 h**

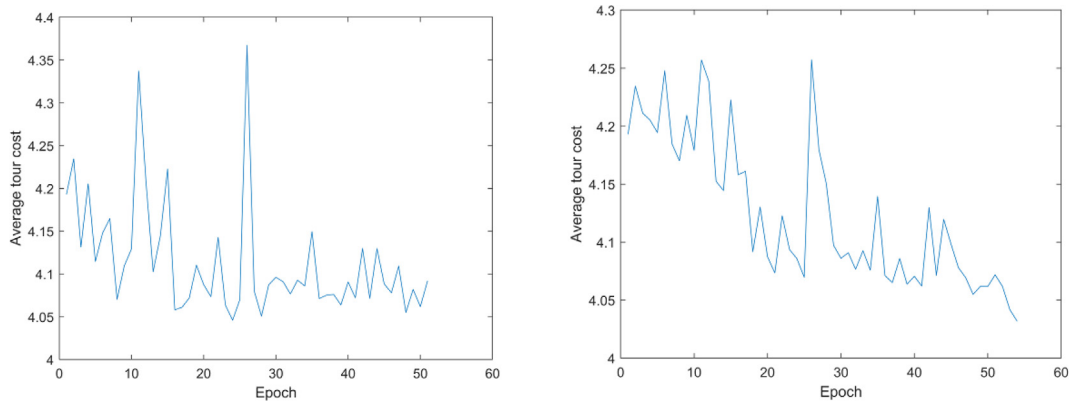

Fig. 5. The learning curves of BDRL on TSP20.

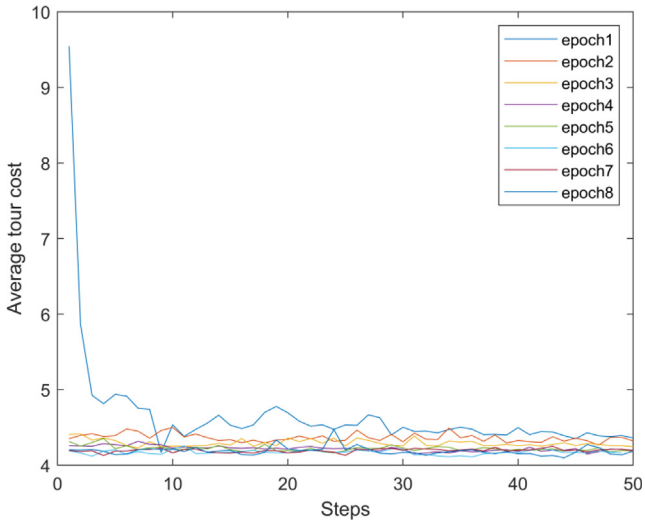

Fig. 6. The average tour cost in the first eight epochs on TSP20.

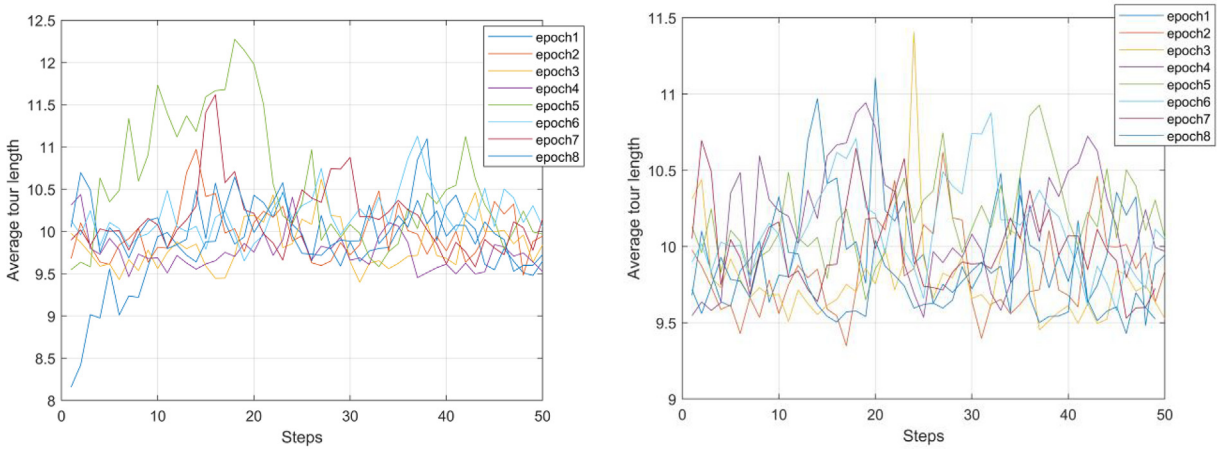

Fig. 7. The performance of the TSP100 pre-trained model on a new TSP100 data set.

nodes. As with TSP20, we examine the change in the average route length obtained over the first eight epochs of BDRL as the time step increases. The experimental results in Fig. 7 (the left figure shows the test curves without CL, and the right figure shows the test curves with CL) show that the learning curves fluctuate more than on TSP20 because the new data distribution differs from the data distribution during training. The solutions obtained by the learning curve with CL are concentrated between 9.4 and 11.5, which is significantly more stable and robust than the learning curve without CL.

In contrast to training and testing on similar datasets, transferring a trained model to new data will cause the agent to adapt to the new environment when testing on the new data due to the relative inconsistency of the data distribution. Therefore, it is reasonable that the agent's rewards become unstable in the initial stage, indicating that the agent is trying to adapt to the variability of the data. We also see an apparent trend that the learning curves gradually stabilize and converge as the number of iterations increases, demonstrating our approach's effectiveness during testing.

4.5. Discussion

The experimental results show that BDRL is superior to the baseline methods in terms of solution accuracy, generalization, and learning and reasoning efficiency. In particular, BDRL can achieve higher quality solutions on many problem instances of different sizes and types in less time. As mentioned above, reinforcement learning training (especially when combined with large-scale deep learning models) is usually unstable and inefficient. Still, the stability and fast convergence of BDRL during training and testing show that our hybrid method is effective in modeling and training. Through thorough comparative analysis, we believe that a combination of factors determines improved performance. Specifically, compared with the previous methods, we replace the original transformer with BERT and use the BERT mask to introduce contrastive learning to improve the data sampling efficiency of reinforcement learning and the training efficiency of the transformer.

Further, we use ReZero and multi-head attention to simplify the transformer's encoder to handle context information more efficiently. We propose a novel hybrid method compared to previous methods based on construction or improvement heuristics to achieve a balance between fast solution generation and guaranteed solution quality. We leverage BERT's pre-training/fine-tuning paradigm to bridge the construction and improvement heuristics, allowing it to generate efficient solutions from scratch and continuously improve the quality of the generated solutions. However, BDRL also has limitations. Although we use hierarchical reinforcement learning to optimize the training process, it still has more parameters than the original transformer.

5. Conclusion and future work

This paper integrates BERT and reinforcement learning to propose a novel training paradigm based on pre-training and fine-tuning for combinatorial optimization. It does not require labeled data and upgrades the transformer layer in BERT to better aggregate the node information in the graph, which is more conducive to finding candidate feasible solutions. We create a brand-new contrastive representation learning objective that considers BERT's mask predictions. We can employ the pre-training and fine-tune in BDRL either independently or together. Our experiments prove the efficiency and effectiveness of BDRL in graph combinatorial optimization problems. Theoretically, if we can apply NLP models such as BERT to combinatorial optimization problems, we can also apply GPT [15] to these problems. However, since these models are often too large to be easily fully trained with limited computing resources, we will continue to explore how to make these state-of-the-art models and training methods more lightweight in the future.

CRediT authorship contribution statement

Qi Wang: conceived and designed the study, and conducted experiments, wrote the paper. Kenneth H. Lai: Helped review and edit the paper and contributed substantially to its publication. Chunlei Tang: conceived and designed the study, and conducted experiments, reviewed and edited the manuscript. All authors read and approve the manuscript.

Data availability

Data will be made available on request.

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.

Acknowledgments

The authors thank Yangyong Zhu, Ph.D., and Yun Xiong, Ph.D., for valuable comments and suggestions on the early versions. The content is solely the responsibility of the authors.

References

[1] T. Bachlechner, B.P. Majumder, H.H. Mao, G.W. Cottrell, J. McAuley, ReZero is All You Need: Fast Convergence at Large Depth, (2020) 1–11.
[2] I. Bello, H. Pham, Q.V. Le, M. Norouzi, S. Bengio, Neural combinatorial optimization with reinforcement learning, in: 5th Int. Conf. Learn. Represent. ICLR 2017 - Work. Track Proc., 2019, pp. 1–15.
[3] Q. Cappart, T. Moisan, L.-M. Rousseau, I. Prémont-Schwarz, A. Cire, Combining Reinforcement Learning and Constraint Programming for Combinatorial Optimization, Thirty-Fifth AAAI Conf. Artif. Intell. Parameterizing, 2021.
[4] T. Chen, S. Kornblith, M. Norouzi, G. Hinton, A simple framework for contrastive learning of visual representations, in: 37th Int. Conf. Mach. Learn. ICML 2020. PartF16814, 2020, pp. 1575–1585.
[5] X. Chen, Y. Tian, Learning to perform local rewriting for combinatorial optimization, Adv. Neural Inf. Process. Syst., 2019.
[6] P.R.de O. da Costa, J. Rhuggenaath, Y. Zhang, A. Akcay, Learning 2-opt Heuristics for the Traveling Salesman Problem via Deep Reinforcement Learning, Asian Conf. Mach. Learn. (2020) 465–480.
[7] H. Dai, E.B. Khalil, Y. Zhang, B. Dilkina, L. Song, Learning combinatorial optimization algorithms over graphs, Adv. Neural Inf. Process. Syst. (2017) 6349–6359.
[8] J. Devlin, M.W. Chang, K. Lee, K. Toutanova, Bert,, Pre-training of deep bidirectional transformers for language understanding, in: NAACL HLT 2019–2019 Conf. North Am. Chapter Assoc. Comput. Linguist. Hum. Lang. Technol. - Proc. Conf., 2019, pp. 4171–4186.
[9] M.Á. Domínguez-Ríos, F. Chicano, E. Alba, Effective anytime algorithm for multiobjective combinatorial optimization problems, Inf. Sci. (Ny) 565 (2021) 210–228.
[10] L. Duan, Y. Zhan, H. Hu, Y. Gong, J. Wei, X. Zhang, Y. Xu, Efficiently Solving the Practical Vehicle Routing Problem: A Novel Joint Learning Approach, in: Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min., 2020, pp. 3054–3063.
[11] M. Fortunato, M. Tan, R. Faulkner, S. Hansen, A.P. Badia, G. Buttimore, C. Deck, J.Z. Leibo, C. Blundell, Generalization of reinforcement learners with working and episodic memory, Adv. Neural Inf. Process. Syst. (2019).
[12] R. Gama, H.L. Fernandes, A reinforcement learning approach to the orienteering problem with time windows, Comput. Oper. Res. 133 (2021) 105357.
[13] A. General, G. National, ALBERT..Parks for S a l e, Order A J. Theory Ordered Sets Its Appl. (2020) 1–16.
[14] J.B. Grill, F. Strub, F. Altché, C. Tallec, P.H. Richemond, E. Buchatskaya, C. Doersch, B.A. Pires, Z.D. Guo, M.G. Azar, B. Piot, K. Kavukcuoglu, R. Munos, M. Valko, Bootstrap your own latent a new approach to self-supervised learning, Adv. Neural Inf. Process. Syst. (2020).
[15] X. Han, Z. Zhang, N. Ding, Y. Gu, X. Liu, Y. Huo, J. Qiu, Y. Yao, A. Zhang, L. Zhang, W. Han, M. Huang, Q. Jin, Y. Lan, Y. Liu, Z. Liu, Z. Lu, X. Qiu, R. Song, J. Tang, J.R. Wen, J. Yuan, W.X. Zhao, J. Zhu, Pre-trained models: Past, present and future, AI Open. 2 (2021) 225–250.
[16] K. He, H. Fan, Y. Wu, S. Xie, R. Girshick, Momentum Contrast for Unsupervised Visual Representation Learning, Proc. IEEE Comput. Soc. Conf. Comput. Vis Pattern Recognit. (2020) 9726–9735.
[17] M. Hutsebaut-Buysse, K. Mets, S. Latré, Hierarchical Reinforcement Learning: A Survey and Open Research Challenges, Mach. Learn. Knowl. Extr. 4 (2022) 172–221.
[18] M. Jaderberg, V. Mnih, W.M. Czarnecki, T. Schaul, J.Z. Leibo, D. Silver, K. Kavukcuoglu, Reinforcement learning with unsupervised auxiliary tasks, ICLR 2017 (2017) 1–17.
[19] Y. Jiang, Y. Wu, Z. Cao, J. Zhang, Learning to Solve Routing Problems via Distributionally Robust Optimization, Proc. AAAI Conf. Artif. Intell. 36 (2022) 9786–9794.
[20] W. Kool, H. Van Hoof, M. Welling, Attention, learn to solve routing problems!, in: 7th Int Conf. Learn. Represent. ICLR 2019, 2019, pp. 1–25.
[21] Y.D. Kwon, J. Choo, B. Kim, I. Yoon, Y. Gwon, S. Min, Pomo,, Policy optimization with multiple optima for reinforcement learning, Adv. Neural Inf. Process. Syst. (2020).
[22] M. Laskin, A. Srinivas, P. Abbeel, CURL: Contrastive unsupervised representations for reinforcement learning, in: 37th Int. Conf. Mach. Learn. ICML 2020. PartF16814, 2020, pp. 5595–5606.
[23] S. Li, Z. Yan, C. Wu, Learning to Delegate for Large-scale Vehicle Routing, Adv. Neural Inf. Process. Syst. 31 (2021) 26198–26211.
[24] J. Long, Z. Sun, P.M. Pardalos, Y. Hong, S. Zhang, C. Li, A hybrid multi-objective genetic local search algorithm for the prize-collecting vehicle routing problem, Inf. Sci. (Ny) 478 (2019) 40–61.
[25] S. Lu, Hao, Zhang, Xingwen, Yang, A learning-based iterative method for solving vehicle routing problems, Iclr 2020. 3 (2018) 1–13.
[26] Q. Ma, S. Ge, D. He, D. Thaker, I. Drori, Combinatorial Optimization by Graph Pointer Networks and Hierarchical Reinforcement Learning, ArXiv. (2019).
[27] S. Manchanda, A. Mittal, A. Dhawan, S. Medya, S. Ranu, A. Singh, Learning Heuristics over Large Graphs via Deep Reinforcement Learning, Assoc. Adv. Artif. Intell. (2019).
[28] J. Mitrovic, B. McWilliams, J. Walker, L. Buesing, C. Blundell, Representation Learning via Invariant Causal Mechanisms, ICLR 2021 (2021) 1–12.
[29] M. Nazari, A. Oroojlooy, M. Takáč, L.V. Snyder, Reinforcement learning for solving the vehicle routing problem, Adv. Neural Inf. Process. Syst. (2018) 9839–9849.
[30] Y. Niu, J. Shao, J. Xiao, W. Song, Z. Cao, Multi-objective evolutionary algorithm based on RBF network for solving the stochastic vehicle routing problem, Inf. Sci. (Ny) 609 (2022) 387–410.
[31] A. Noormohammadi-Asl, H.D. Taghirad, Multi-goal motion planning using traveling salesman problem in belief space, Inf. Sci. (Ny) 471 (2019) 164–184.
[32] E. Parisotto, H.F. Song, J.W. Rae, R. Pascanu, C. Gulcehre, S.M. Jayakumar, M. Jaderberg, R.L. Kaufman, A. Clark, S. Noury, M.M. Botvinick, N. Heess, R. Hadsell, Stabilizing transformers for reinforcement learning, in: 37th Int. Conf. Mach. Learn. ICML 2020. PartF16814, 2020, pp. 7443–7454.
[33] R. Qi, J. Qing Li, J. Wang, H. Jin, Y. Yan Han, QMOEA: A Q-learning-based multiobjective evolutionary algorithm for solving time-dependent green vehicle routing problems with time windows, Inf. Sci. (Ny). 608 (2022) 178–201.
[34] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, Adv. Neural Inf. Process. Syst. (2017) 5999–6009.
[35] O. Vinyals, M. Fortunato, N. Jaitly, Pointer networks, Adv. Neural Inf. Process. Syst. (2015) 2692–2700.
[36] E. Voita, R. Sennrich, I. Titov, The bottom-up evolution of representations in the transformer: A study with machine translation and language modeling objectives, in: EMNLP-IJCNLP 2019 - 2019 Conf. Empir. Methods Nat. Lang. Process. 9th Int. Jt. Conf. Nat. Lang. Process. Proc. Conf., 2019, pp. 4396–4406.
[37] F. Wang, X. Wang, S. Sun, A reinforcement learning level-based particle swarm optimization algorithm for large-scale optimization, Inf. Sci. (Ny) 602 (2022) 298–312.
[38] Q. Wang, C. Tang, Deep reinforcement learning for transportation network combinatorial optimization: A survey, Knowledge-Based Syst. 233 (2021) 107526.
[39] H. Woo, H. Lee, S. Cho, An Efficient Combinatorial Optimization Model Using Learning-to-Rank Distillation, Proc. AAAI Conf. Artif. Intell. 36 (2022) 8666–8674.
[40] Y. Wu, W. Song, Z. Cao, J. Zhang, A. Lim, Learning Improvement Heuristics for Solving Routing Problems, IEEE Trans. Neural Networks Learn. Syst. 33 (2022) 5057–5069.
[41] Z. Wu, Q. Gao, B. Jiang, H.R. Karimi, Solving the production transportation problem via a deterministic annealing neural network method, Appl. Math. Comput. 411 (2021) 126518.
[42] Z. Wu, H.R. Karimi, C. Dang, An approximation algorithm for graph partitioning via deterministic annealing neural network, Neural Networks. 117 (2019) 191–200.
[43] L. Xin, W. Song, Z. Cao, J. Zhang, Multi-Decoder Attention Model with Embedding Glimpse for Solving Vehicle Routing Problems, in: 35th AAAI Conf. Artif. Intell. AAAI 2021, 2021, pp. 12042–12049.

[44] L. Xin, W. Song, Z. Cao, J. Zhang, NeuroLKH: Combining Deep Learning Model with Lin-Kernighan-Helsgaun Heuristic for Solving the Traveling Salesman Problem, Adv. Neural Inf. Process. Syst. (2021) 7472–7483.
[45] F. Yin, Y. Zhao, Distributionally robust equilibrious hybrid vehicle routing problem under twofold uncertainty, Inf. Sci. (Ny) 609 (2022) 1239–1255.
[46] I.A. Zamfirache, R.E. Precup, R.C. Roman, E.M. Petriu, Policy Iteration Reinforcement Learning-based control using a Grey Wolf Optimizer algorithm, Inf. Sci. (Ny) 585 (2022) 162–175.
[47] J. Zhang, H. Zhang, C. Xia, L. Sun, Graph-Bert: Only Attention is Needed for Learning Graph Representations, ArXiv. (2020).
[48] J. Zheng, K. He, J. Zhou, Y. Jin, C.-M. Li, Combining Reinforcement Learning with Lin-Kernighan-Helsgaun Algorithm for the Traveling Salesman Problem, Assoc. Adv. Artif. Intell. (2020).
[49] Z. Zong, H. Wang, J. Wang, M. Zheng, Y. Li, RBG: Hierarchically Solving Large-Scale Routing Problems in Logistic Systems via Reinforcement Learning, Association for Computing Machinery, 2022.