Applied Intelligence
https://doi.org/10.1007/s10489-021-02920-3

VARL: a variational autoencoder-based reinforcement learning Framework for vehicle routing problems

Qi Wang[1]

Received: 10 May 2021 / Accepted: 9 October 2021

Abstract
The vehicle routing problem as a classic NP-hard problem could be optimized by path choices due to its practical application value. This study proposes a novel variational autoencoder framework for path optimization on graphs, involving graph neural networks and generative adversarial networks. We took the center node as the root node to divide the graph into different subgraphs and find the nodes that compose the optimal solution through variational reasoning. We next used reinforcement learning to optimize the entire variational framework end-to-end. This contribution can also apply in both modeling and training combinatorial optimization over graphs. An extensive experiment on different scales of traveling salesman and vehicle routing instances was conducted. The findings indicate that our framework is efficient and effective in learning and reasoning, and its accuracy and generalization outperform the baselines.

Keywords NP-hard problems · Machine learning · Reinforcement learning · Variational autoencoders · Variational reasoning

1 Introduction

Combinatorial optimization, such as the traveling salesman problem (TSP) [1] and the vehicle routing problem (VRP) [2], has been studied for long periods. Recent research on its traditional methods [3] (e.g., exact algorithms, approximation algorithms, and heuristic algorithms [4], etc.) failed to take advantage of the fact that most combinatorial optimization problems have similar internal structures and are distinguished only by data and variables [5]. In many applications, the coefficients of the objective function or constraint are sampled from the same distribution. I hope to get a general method to improve the efficiency and quality of problem-solving, and the application of machine learning [6] (including deep learning [7], reinforcement learning [8], and so on) to combinatorial optimization should be a promising direction from the current research trend [9].

Compared with the traditional optimization approach for only one task, machine learning can automatically discover features through training data, requiring less manual label and expert experience. The model is more generalizable and suitable for many optimization tasks. Deep reinforcement learning is a new research field in machine learning, which combines the perceptual ability of deep learning with the decision-making ability of reinforcement learning, and realizes direct control from original input to output through end-to-end learning. Deep learning combines low-level features to form more abstract high-level representation attributes, categories, or features through deep network structure and nonlinear transformation to discover distributed feature representation of data, which focuses on the perception and expression of things. Reinforcement learning learns the optimal policy to maximize the agent's accumulative reward from the environment and learn the policy to solve the problem. Faced with increasingly complex real-world tasks, researchers propose using input data representations as a basis for self-improving reinforcement learning and then combining deep learning and reinforcement learning to propose deep reinforcement learning. In recent years, deep reinforcement learning has led to a revolutionary breakthrough in the field of artificial intelligence [10].

At present, there are still problems in the combination optimization method based on machine learning. For example, it is difficult for the RNN model to use the structural information in the graph, so the logical reasoning ability is

✉ Qi Wang
17110240039@fudan.edu.cn

[1] Shanghai Key Laboratory of Data Science, School of Computer Science, Fudan University, Shanghai, China

Published online: 09 November 2021

weak, especially when the target node is not the direct neighbor of the current node. It needs to be on the graph to perform logical reasoning to infer multi-hop neighbor nodes. In the training stage, although reinforcement learning improves the model's generalization ability and solves the problem of requiring a large amount of expensive label data, it is still not as accurate as supervised learning in terms of training. Reinforcement learning has sparse reward problems and insufficient exploration space, especially in the early stage of training [11]. Some researchers have applied a two-stage training framework [12, 13] (supervised learning followed by reinforcement learning) for the warm start. Still, compared with the end-to-end paradigm, multi-stage training is likely to lead to the cascade transmission of errors. The previous training will pass the errors to the later training, which may be infinitely amplified. In addition, it is a challenge to select an appropriate reinforcement learning algorithm based on applied neural network modeling (continuous or discrete actions).

The main methods include Monte Carlo Markov Chain (MCMC) and Variational Inference (VI) are proposed to solve reasoning. Since each step of MCMC training requires large quantities of data, the training cost is very high, while the variational inference can be trained with BP algorithm and small-batch gradient descent, so the cost is lower. The VAE is precisely on the development of variational reasoning. It inherits the idea of variational inference to use a distribution to approximate the posterior distribution [14]. The difference is that VAE considers the posterior distribution of all data simultaneously and approximates each posterior distribution with distribution, minimizing KL divergence. The VAE successfully applies neural networks to inference problems and solves the problem of continuous data generation. In reasoning, it has the advantages of fast training and low cost. The generation problem has the advantages of fast training, high stability, high diversity, and so on [15]. In this paper, VRP is modeled as a variational probability problem, consisting of two parts: prior and posterior probabilities. We use the variational graph message network as the graph encoder, take the graph as input, and update the nodes and edges. I identify the central node in the graph and use it as the root node to divide the whole graph into different subgraphs. We propose the graph inference network to determine the feasible solution composed of the sequence of feasible nodes on each subgraph. We incorporated the GAN into variational learning to make the optimization process more efficient and robust and utilized REINFORCE with variance reduction to train the whole framework.

In summary, the contributions of this paper are as follows:

- We are the first to employ the paradigm of variational autoencoder to reason VRP instances on graphs, which is more efficient, requires fewer label data, and can be learned end-to-end.
- We identify the central node in the graph and use it as the root to divide it into different subgraphs, equivalent to transforming VRP into TSP. We can simultaneously solve VRP and TSP and propose a novel graph inference network to reason subgraphs.
- In the optimization process of variational learning, we integrate the GAN to make the whole framework more robust and efficient. Moreover, we combine REINFORCE and variance reduction to train the framework suitable for the variational reasoning framework.

The rest of this paper is organized as follows. Section 2 presents the related works of the paper. Section 3 describes the proposed method and gives the details of its components. Section 4 depicts the experimental results and relevant analysis. Finally, Section 5 concludes the paper.

2 Literature review

The application of machine learning to combinatorial optimization (CO) has gradually become a research hotspot in recent years [16]. Especially the success of deep learning and reinforcement learning in graph data makes it possible to become another critical milestone in solving CO problems [9].

Neural networks for modeling based on machine learning can be roughly divided into two types, namely models based on RNN [17] or attention [18] and models based on graph neural network (GNN) [19]. Moreover, their training methods can be divided into supervised learning and unsupervised learning, in which reinforcement learning is the mainstream at present [20]. Next, I will introduce some representative works in recent years.

RNN or attention-based models Applying neural networks in CO can be traced back to the Hopfield network [21], used to solve small-scale TSP. RNN is designed to handle sequence information and can handle associations between input variables. The encoder-decoder [22] is a kind of general framework to solve the sequence-to-sequence problem. The pointer network [23] is a neural architecture based on a sequence-to-sequence network used to learn the conditional probability of the output sequence. The elements are discrete markers corresponding to the positions in the input sequence. In general, the content of the output sequence of the combinatorial optimization problem is the same as the content of the input sequence. Still, the order of the sequence is changed. Moreover, solving the CO problem involves making sequence decisions, so the pointer network is a very

targeted neural network architecture for the combinatorial optimization problem.

Bello et al. took the lead in trying to employ reinforcement learning [24] (actor-critic framework [25]) to train pointer networks to solve combinatorial optimization problems [26]. They explained that combinatorial optimization problems often lack labeled data, and supervised learning with enough labeled data cannot be generalized as reinforcement learning only from experience and intuition.

Nazari et al. proposed an end-to-end framework for solving vehicle routing problems using reinforcement learning based on pointer networks and [26], which can handle any problem sampling from a given distribution, rather than training individual models for each problem instance [27]. I can apply the framework if it approximates the generation distribution, an extension of [26], and is more generalizable. I can treat the model as a black box heuristic (or meta-algorithm) that generates high-quality solutions in a reasonable amount of time.

Kool et al. used a deterministic greedy rollout as a baseline to guide the improved transformer [18] to learn the CO algorithm [28]. The attention model they proposed also consists of an encoder and a decoder, in which the encoder uses a graph attention network (GAT) [29] with input as the coordinate of each node and output as the representation of each node, and the graph representation is the average embedded value of all nodes. The decoder also uses GAT, and its input is a combination of node embedding, context embedding, start node embedding, and previous node embedding. The output is a series of nodes with the highest compatibility, then selected at each step to be added to the path. They employ the mask to ensure the viability of the solution.

GNN-based models Khalil et al. took the lead in using the combination of reinforcement learning and GNN (structure2vec) to solve combinatorial optimization problems on graphs [5]. The policy it learns resembles a meta-algorithm that incrementally constructs a solution, and a graph embedding determines the actions on the current state of the solution. Based on the graph structure, the feasible solution is constructed by continuously greedily adding nodes, and the feasible solutions are kept satisfying the graph constraint of the problem.

Li et al. applied GCN [30] with supervised learning and tree search to solve NP-hard problems [31]. Many classic methods, such as local search and graph reduction, were integrated into the deep learning framework, making it elegant to use traditional deep learning to enable traditional methods to solve classic problems. Based on [5] and [31], Mittal et al. first applied the GCN to embed and aggregate graph data and then applied supervised learning followed by reinforcement learning to train the model [13]. Besides, they employed a greedy probability distribution to preprocess graphs so that the model could deal with large-scale graphs.

Ma et al. improved the pointer network into a GNN, namely graph pointer network, combining the advantages of both pointer network and GNN [32]. They applied hierarchical reinforcement learning to train GNN in layers to make model training more efficient. Duan et al. applied attention and GCN to model the VRP jointly and applied supervised learning and reinforcement learning to train the whole framework [12]. They made GCN into the encoder-decoder paradigm and used the pointer for sequence prediction and edge classification, making the framework handle graph and edge. There are similar recent works that combine GNN and reinforcement learning for combinatorial optimization, such as [33–37], some of which also combine the attention mechanism.

Besides, inspired by AlphaGo Zero [38], MuZero [39], etc., there are now scholars who integrate GNN, reinforcement learning, and Monte Carlo tree search to solve combinatorial optimization problems through self-play [40–42]. They take advantage of the fact that AlphaGo Zero does not require expert experience. Still, these methods have a common problem: they require too much computing resources and time to support model training, which is not sustainable under standard experimental conditions.

Variational auto-encoder With the rapid development of machine learning [21], many complex scenarios have attracted people's attention in recent years. In these scenarios, inference and training are often complex and costly [43]. On the one hand, many classical algorithms often have some difficult conditions or too strong constraints during inference and training, so these algorithms cannot meet the needs of complex scenes [44]. On the other hand, due to the back-propagation (BP) invention and maturity, small-batch gradient descent has become an efficient method with low training costs [45]. Therefore, a natural development requirement is an inference model that can be applied to complex scenarios and trained with small batches of gradient descent. VAE is the natural development of variational inference, which combines the advantages of the ELBO and neural networks to solve inference in general scenarios and the problem of continuous data generation [14]. It has many advantages, including fast training, stability, and so on, so it has a wide range of theoretical models and industry applications.

3 Methodology

This section describes the proposed method in detail. We first formally define the problem studied in Section 3.1 and give the background knowledge of the proposed method. We

describe the proposed variational probability model for VRP in Section 3.2, which includes three parts: the graph encoder, the graph reasoning network, and the graph decoder. Then in Section 3.3, we give the objective function of end-to-end variational learning based on the previous graph neural network. Finally, Section 3.4 proposed a novel reinforcement learning algorithm for VRP.

3.1 Problem definition and background

A VRP instance can be represented by a directed fully-connected graph $G = (V, E)$ with $V = \{0, \ldots, n\}$, and $E = \{e_{ij}\}$, where node $i = 0$ denotes the warehouse or the central node, and $i \in \{1, \ldots, n\}$ are the customer nodes, $e_{ij}(i, j \in V)$ denotes the set of edges between nodes. Each node is associated with a feature vector x_i^d, and each edge is associated with a feature vector d'_{ij}, where x_i^d represents the demand of nodes and d'_{ij} represents the distance between two nodes.

Suppose there is a warehouse with k vehicles, the vehicle capacity is Q, and each customer has a demand x_i^d. The vehicles start from the warehouse to deliver services to customers and then return to the warehouse. All customers must be delivered, and each customer is delivered once, and the vehicle capacity limit cannot be violated. The goal is to minimize the total distance of all vehicle routes. In our model, we aim to find a customer node sequence $\boldsymbol{\pi} = \{\pi_1, \pi_2, \pi_3, \ldots, \pi_T\}$, where $\pi_t \in \{v_0, v_1, v_2, \ldots, v_n\}$, $v_0 = 0$ may occur multiple times, but other nodes can only occur once, so the sequence between two "0" is the vehicle's path. If there are only two "0" in the sequence, it becomes a TSP where only one vehicle traverses all customer nodes.

The objective function is as follows:

$$minck + c' \sum_{t=1}^{T-1} d'_{\pi_t, \pi_{t+1}} \tag{1}$$

Where c is the fixed cost of a vehicle (including fuel fee, driver's remuneration, etc.), c' is the unit cost of running.

Variational autoencoders VAE results from the combination of variational inference and neural networks. We first introduce VAE's ideological background: variational inference and ELBO. Typical data such as pictures, videos, audio, etc., we often assume that it is generated by some lower-level variables that satisfy certain distributions, called latent variables. These latent variables represent the internal structure or abstraction of the data. Set the data variable as the first and the latent variable as z, and then the general assumption is the following generation model:

$$p(x, z) = p(z)p(x|z) \tag{2}$$

Where $p(z)$ is the distribution of latent variables, known as prior distribution; $p(x|z)$ is assumed to be a specific distribution, such as the Gaussian distribution. This assumption enables the marginal distribution $p(x)$ to fit the arbitrarily smooth data distribution $q(x)$.

A paradigm of the inference problem: Given a data $x^{(i)}$ and assuming that the training has been completed such that $p(x) = q(x)$, how can the posterior distribution $p(z|x^{(i)})$ be inferred? According to the Bayesian formula, we can get:

$$p\left(z|x^{(i)}\right) = \frac{p(z)p\left(x^{(i)}|z\right)}{p\left(x^{(i)}\right)} \tag{3}$$

Variational inference and ELBO The variational inference approximates the posterior distribution $p(z|x^{(i)})$ with a distribution $q(z)$, specifically by minimizing the following KL divergence:

$$KL(q(z)\|p(z|x^{(i)})) = E_{q(z)}[logq(z)] - E_{q(z)}[logp(z, x^{(i)})] + logp(x^{(i)}) \tag{4}$$

Since $logp\left(x^{(i)}\right)$ is an unknown constant, and we can directly maximize the negative formula as follows:

$$ELBO(q(z)) = E_{q(z)}\left[logp\left(z, x^{(i)}\right)\right] - E_{q(z)}\left[logq(z)\right] \leq logp\left(x^{(i)}\right) \tag{5}$$

The above expression is called ELBO (the Evidence Lower Bound), the lower bound of log-evidence.

ELBO enables us to unify the training of the generation model and the inference model. When training the generation model, maximizing the likelihood function is equivalent to maximizing ELBO, and when training the inference model, minimizing KL divergence is also equivalent to maximizing ELBO.

VAE inherited the idea of variational inference to approximate the posterior distribution with a distribution. The difference is that VAE considers the posterior distribution of all data at the same time and approximates each posterior distribution with distribution, that is, minimizes the following KL divergence:

$$\sum_{i=1}^{N} KL\left(q\left(z|x^{(i)}\right)\middle| |p\left(z|x^{(i)}\right)\right) = -L + \sum_{i=1}^{N} logp\left(x^{(i)}\right) \tag{6}$$

Where L is the lower bound of the likelihood function, and its specific form is:

$$L = \sum_{i=1}^{N} E_{q(z|x^{(i)})}[-logq\left(z|x^{(i)}\right) + logp\left(z|x^{(i)}\right)] \leq \sum_{i=1}^{N} logp\left(x^{(i)}\right) \tag{7}$$

3.2 The variational probabilistic model for VRPs

Previous machine learning-based methods [26, 28] typically define a stochastic policy $p(\boldsymbol{\pi}|s)$ for selecting the

solution set $\boldsymbol{\pi}$ on a given problem instance s, and it is parameterized by θ as:

$$p_\theta(\boldsymbol{\pi}|s) = \prod_{t=1}^{n} p_\theta\left(\pi_t|s, \boldsymbol{\pi}_{1:t-1}\right) \tag{8}$$

The core idea of applying machine learning in combinatorial optimization on the graph is to learn the probability distribution of solutions through a given problem instance. In our probability model, according to Eqs. (2) and (3), we model the compatibility between solutions and instances as a probability model $P_{\theta_1}\left(\pi_t|s\right)$, which represents the probability of a solution π_t being obtained by a given instance s in the graph.

We propose a novel propagation-like deep learning architecture on the graph to perform logical reasoning in a probabilistic model. Given an instance s and its latent variables π_t, we need to reason on the graph to get the correct node v_i on the route π_t, specifically, we need to simulate a sequence of nodes v_i in the given solution π_t and the possibility $P_{\theta_2}\left(v_i|\pi_t, s\right)$ of instance s, as the solution π_t in the instance s is not marked, so we naturally formulate a problem instance s by regarding the solution π_t as a latent variable. The probability that the node v_i found in the simulation instance s is correct can be obtained by summing up all the possibilities of the latent variable:

$$\sum\nolimits_{v_i \in V} P_{\theta_1}\left(\pi_t|s\right) P_{\theta_2}\left(v_i|\pi_t, s\right) \tag{9}$$

Given a training data set D_{train} consisting of N optimal paths, the parameters are θ_1 and θ_2, and the objective function is:

$$max_{\theta_1,\theta_2} \frac{1}{N} \sum\nolimits_{i=1}^{N} log\left(\sum\nolimits_{\boldsymbol{\pi} \in V} P_{\theta_1}\left(\pi_t|s\right) P_{\theta_2}\left(v_i|\pi_t, s\right)\right) \tag{10}$$

Graph encoder Combinatorial optimization is usually based on the encoder-decoder paradigm because we can obtain the intrinsic structural features of the graph through the encoder and operate the graph by the decoder, such as reasoning and reconstruction. We first apply a graph messaging passing network [46] to encode a potential representation of G, where each vertex v has a feature vector x_v to denote the properties of the node. Similarly, each edge e has a feature vector x_e to maintain its information and the two hidden vectors v_{ij} and v_{ji} represent information from the node v_i to v_j, and vice versa. Because of the loopy structure of graphs, the message is exchanged in a loopy belief propagation mode:

$$v_{ij}^{(t)} = \tau(W_1^g x_v + W_2^g x_e + W_3^g \sum\nolimits_{w \in N(i) \setminus j} v_{wi}^{(t-1)}) \tag{11}$$

Where $v_{ij}^{(t)}$ is the message calculated in the $t-th$ iteration, which is initialized with $v_{ij}^{(0)} = 0$, $N(i)$ represents a set of neighbor nodes of a node v_i. After iteration in T steps, we aggregate the message into a latent vector for each vertex, which captures its local structure of the graph:

$$h_i = \tau(U_1^g x_e + \sum\nolimits_{j \in N(i)} U_2^g v_{ij}^{(T)}) \tag{12}$$

$$h_G = \sum\nolimits_i h_i / |V| \tag{13}$$

Where h_G is the final graph representation, the log variance $log\sigma_G$ of the variational posterior approximation and the mean μ_G are computed from h_G with two separate affine layers, z_G is sampled from the Gaussian distribution $N(\mu_G, \sigma_G)$.

In VRP, no matter how other variables change, the central node is the core of the VRP and always exists objectively, and it is not identified on the graph or in the training data. If we take the graph as input, we feel it is necessary to identify the center node because it marks the beginning and end of all paths, but previous works often know the center node by default and only match the center node with the "0" marker. It is necessary to deal with instances and central nodes on the graph to search and reason about each path from the global and local information of the graph.

More specifically, previous works did not do anything with problem instances, and the mapping from instances to solutions relied on the corresponding expertise and heuristics. However, in the practical application, the ideal situation should be given a problem instance to get the corresponding solution to achieve end-to-end learning and optimization. Our framework is instance-oriented, creating an automatic mapping from instance to solution (Fig. 1). For example, if we input the above VRP instance, our framework should identify the theme and find and reason the optimal solution on the graph. If we input a variant of VRP, such as the TSP instance, the framework will be fine-tuned accordingly.

Since the central node in the D_{train} is not marked, the node that needs to be identified will be treated as a latent variable. The specific steps to identify the subject are as follows: (1) The input instance s is first encoded then converted into a vector with dimension d by using the neural network. (2) Convert every node in the graph into a vector. (3) "*Softmax*" is used to calculate the probability that each node in the graph under s is a node in the instance. Specifically:

$$P_{\theta_1}\left(\pi_t|s\right) = softmax\left(W_{\pi_i}^T f(s)\right) = \frac{exp\left(W_{\pi_i}^T f(s)\right)}{\sum_{\pi' \in V} exp\left(W_{\pi_i'}^T f\left(s\right)\right)} \tag{14}$$

Where W is the parameter, and $f(\cdot)$ is a neural network that embeds the vector. For example, when the instance is text, RNN can be selected, while CNN can be selected when it is an image. It is more practical for the traditional

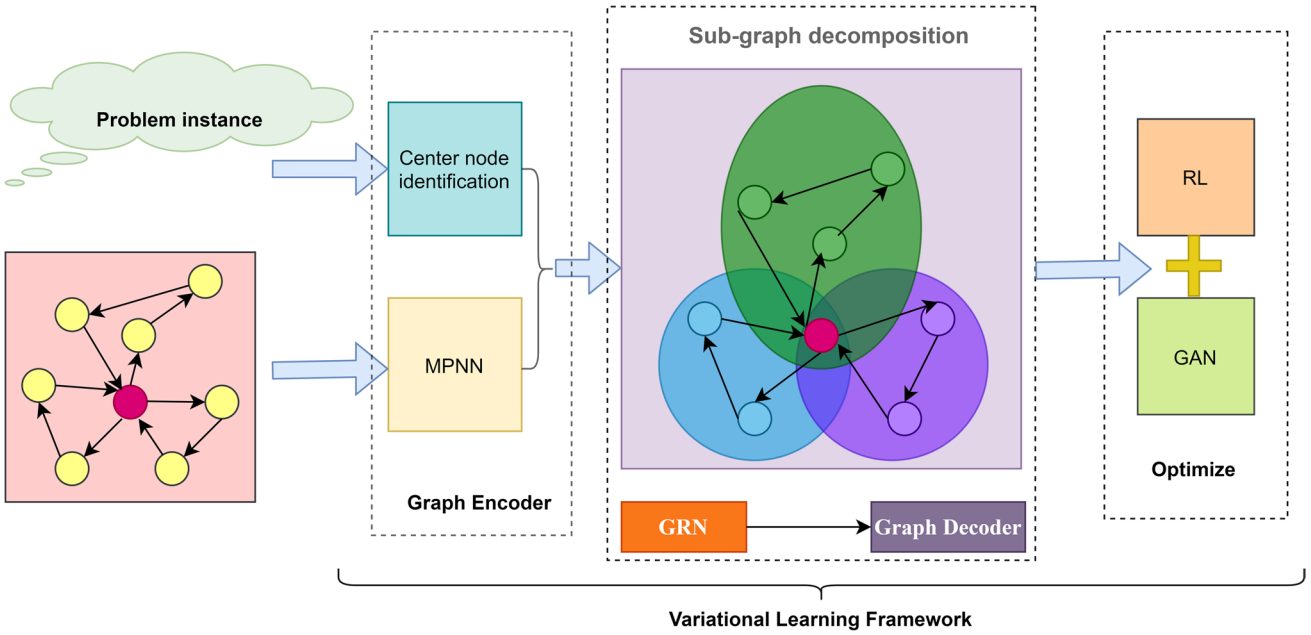

Fig. 1 The complete flow diagram of VARL. Our variational learning framework follows the encoder-decoder paradigm. A graph is taken as input, GNN is taken as graph encoder, and GRN divides the graph into different sub-graphs with the central node as the root and reasoning. Finally, GAN and RL are used to optimize the whole framework

combinatorial optimization method and can be generalized to multiple application scenarios.

Graph reasoning network Applying parameterized inference models is challenging because retrieving and reasoning the optimal path requires multi-step traversal on a large graph. Therefore, we propose a graph reasoning network (GRN) in which all inference rules and their complex combinations are represented as nonlinear embedding in a vector space and are learned. We assume that the model knows the maximum number of logical reasoning steps T, starting from the central node, we perform a topological sort on all nodes in the T hop according to the graph, and then we can get an ordered list of nodes $v_1, v_2, v_3, \ldots, v_T$.

Since there are no labeled reasoning rules in the whole learning process of the framework, the rules (heuristics) used in reasoning will be learned. The whole reasoning process is shown as follows :(1) The instance s is encoded through another network f_G to transform it into a vector with dimension d. (2) Applying GRN, i.e., the potential optimal solution π_t in the given range of G_{π_t}, we express $G_{\pi_t \to v_i}$ as the minimum subgraph, containing all paths traversed from the central node and encodes the central node's adjacent nodes. (3) "*Softmax*" is used to calculate the probability that the solution is the optimal solution of the instance through reasoning. (4) If the reasoning does not reach the maximum number of steps, the second step (2) is returned. The original node adjacent to the center node is converted into the center node to carry out the reasoning. The probability of using GRN $G_{\pi_t \to v_i}$ and range G_{π_t} to calculate the correctness of the path is described below:

$$\begin{aligned} P_{\theta_2}(v_i|\pi_t, s) &= softmax\left(f_G(s)^T g\left(G_{\pi_t \to v_i}\right)\right) \\ &= \frac{exp\left(f_G(s)^T g\left(G_{\pi_t \to v_i}\right)\right)}{\sum_{v_i' \in V} exp\left(f_G(s)^T g\left(G_{\pi_t \to v_i'}\right)\right)} \end{aligned} \tag{15}$$

Where $g\left(G_{\pi_t \to v_i}\right)$ is the proposed GRN, a GNN that resembles forward filtering in the Hidden Markov Model or the Bayesian Network. The graph reasoning representation of π_t is computed recursively using its parent representation:

$$g\left(G_{\pi_t \to v_i}\right) = \frac{1}{\phi(v_i)} \sum_{v_j \in \phi(v_i), (v_j, e, v_i) or (v_i, e, v_j) \in G_{\pi_t}} ReLU(W_1 \times [g\left(G_{\pi_t \to v_j}\right), \vec{e}]) \tag{16}$$

Where $\vec{e}$ is the embedding of edges, v_j represents the parent node of v_i, W_1 is the parameter, and $\phi(v_i)$ counts the number of parent nodes of v_i in G_{π_t}.

Graph decoder Previous work based on GNN to model large-scale graph instances encountered the problem that the model was too heavy to be trained due to too many parameters. Therefore, we must decompose the large-scale graph into several small subgraphs and then use the divide and conquer algorithm idea to process iteratively. In addition, for the vehicle routing problem (VRP), we can divide the graph into subgraphs based on the central node, which is equivalent to transforming VRP into easier TSP so that the learned data distribution is more refined.

Generalization We divide the graph into several subgraphs according to the central node through graph encoding and graph reasoning. Our goal here is to assemble the subgraphs (nodes in the tree) into the original graph. We can regard the subgraph as a tree structure rooted by the central node and assemble a neighborhood graph to follow the order in which the tree itself is decoded. In other words, we first look at the scores of the central node and its neighbors. We then proceed to assemble neighbors and their associated clusters, and so on. Decoding is similar to the encoding step but with different (learned) parameters:

$$\mu_{ij}^{(t)} = \tau(W_1^d x_v + W_2^d x_e + W_3^d \tilde{\mu}_{ij}^{(t-1)}) \tag{17}$$

$$\tilde{\mu}_{ij}^{(t-1)} = \begin{cases} \sum_{w \in N(i) \setminus j} \mu_{wi}^{(t-1)}, \beta_i = \beta_j \\ \hat{m}_{\beta_i,\beta_j} + \sum_{w \in N(i) \setminus j} \mu_{wi}^{(t-1)}, \beta_i \neq \beta_j \end{cases} \tag{18}$$

We extend the model with a message $\hat{m}_{\beta_i,\beta_j}$ via graph encoder, which provides a subgraph-dependent location context for $bond(i,j)$.

3.3 End-to-end learning of VARL

According to the definition of variational reasoning and ELBO as above, we optimize the ELBO and update the objective function (10):

$$\begin{aligned} &max_{\rho,\theta_1,\theta_2} L(\rho,\theta_1,\theta_2) \\ &= \frac{1}{N}\sum_{i=1}^{N} E_{Q_\rho(\pi_t|s,v_i)}[logP_{\theta_1}(\pi_t|s) + logP_{\theta_2}(v_i|\pi_t,s) - logQ_\rho(\pi_t|s,v_i)] \end{aligned} \tag{19}$$

Where $Q_\rho(\pi_t|s,v_i)$ is the variational posterior probability, which can be learned together with the model. The posterior distribution probability can reduce the approximation error. Q_ρ calculates the likelihood of the solution π_t and the target node v_i for instance s. According to GRN, we define the range $G_{v_i \to \pi_t}$ of the target node v_i and the inverse reasoning graph network $g(G_{v_i \to \pi_t})$ to calculate the embedding. The variational posterior probability consists of two parts:

$$Q_\rho(\pi_t|s,v_i) \propto exp(\tilde{W}_{\pi_t}^T \tilde{f}(s) + \tilde{f}_G(s)^T g(G_{v_i \to \pi_t})) \tag{20}$$

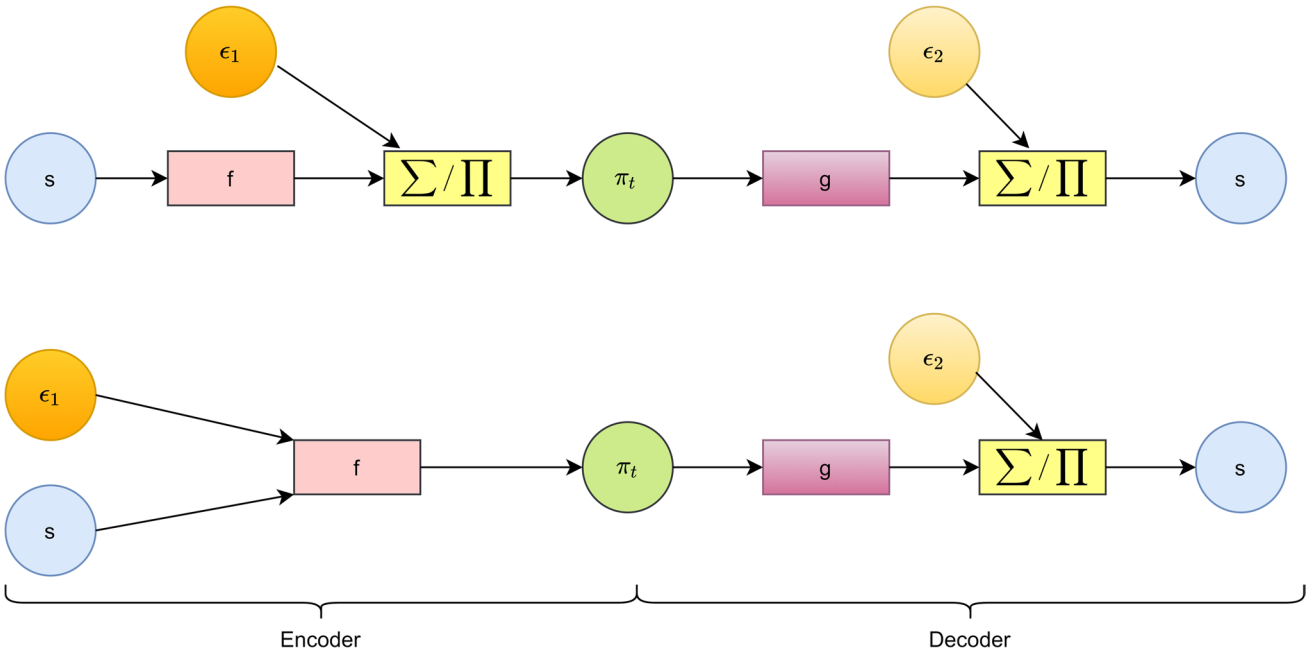

Fig. 2 The difference at the frame level between the standard VAE (above) and the VAE with GAN (below), where ϵ_1 and ϵ_2 denote sampling from the noise distribution. We added noise from the beginning and made the optimization framework simpler to improve the framework's efficiency

Where $\tilde{W}_{\pi_t}^T$, $\tilde{f}_G(s)^T$ are parameters and can be shared with (14), (15).

Since $Q_\rho(\pi_t|s, v_i)$ is usually selected as an easy-to-handle distribution, which limits the flexibility of the model. We introduce the generative adversarial network (GAN) to improve the optimization effect and robustness of the model (Fig. 2). We employ a latent way to express $logP_{\theta_1}(\pi_t|s) - logQ_\rho(\pi_t|s, v_i)$, that is, the adversarial network $T(s, \pi_t)$ is introduced, so that its optimal value is just $logP_{\theta_1}(\pi_t|s) - logQ_\rho(\pi_t|s, v_i)$. Specifically, we transform the optimization problem into the following form:

$$max_T E_{P_{D_{train}(x)}} E_{Q_\rho(\pi_t|s,v_i)} log\sigma\big(T(s,\pi_t)\big) + E_{P_{D_{train}(x)}} E_{p(\pi_t)} \log(1 - \sigma(T(s,\pi_t))) \tag{21}$$

$T(s, \pi_t)$ is applied to determine whether the sample (s, π_t) comes from $P_{D_{train}(x)}Q_\rho(\pi_t|s, v_i)$ or $P_{D_{train}(x)}P(\pi_t)$, according to the optimal discriminant of GAN:

$$T^*(s,\pi_t) = logQ_\rho(\pi_t|s,v_i) - P(\pi_t) \tag{22}$$

At this point, the objective function becomes:

$$max_{\theta_2,\rho} E_{P_{D_{train}(x)}} E_{Q_\rho(\pi_t|s,v_i)}(-T^*(s,\pi_t) + logP_{\theta_2}(v_i|\pi_t,s)) \tag{23}$$

The above optimization objective can easily obtain the gradient for θ_2, but ρ is more troublesome because $T^*(s, \pi_t)$ is related to ρ. However, we have the following equation:

$$E_{Q_\rho(\pi_t|s,v_i)}\big(\nabla_\rho T^*(s,\pi_t)\big) = 0 \tag{24}$$

We apply the re-parametrization to turn the optimization objective into:

$$max_{\theta_2,\rho} E_{P_{D_{train}(x)}} E_\epsilon(-T^*(s,\pi_{t\rho}(s,\epsilon)) + logP_{\theta_2}(s|\pi_{t\rho}(s,\epsilon))) \tag{25}$$

Where ϵ is the noise sampling, and the specific algorithm flow is shown in Algorithm 1.

3.4 REINFORCE with variance reduction

The commonly used reinforcement learning algorithms in previous works include actor-critic algorithm [26], Q-learning [5], REINFORCE [47]. However, it is not easy for the action-critic algorithm to fully play its role, and Q-learning has limited exploration space and cannot be used for continuous action. Kool et al. and Duan et al. use a rollout baseline based on REINFORCE [28, 12], but Monte Carlo sampling requires large quantities of data in each training step. Therefore, we combine REINFORCE and variance reduction [48] to train our variational model to propose a more targeted and efficient reinforcement learning algorithm (REINFORCE with variance reduction).

We assume that there is only one training instance, that is, $N = 1$, and the gradient of ρ with respect to the posterior parameters of L can be calculated as:

$$\nabla_\rho L = E_{Q_\rho(\pi_t|s,v_i)}\big[\nabla_\rho logQ_\rho(\pi_t|s,v_i)\omega(\pi_t,s,v_i)\big] \tag{26}$$

$$\omega(\pi_t,s,v_i) = \log P_{\theta_1}(\pi_t|s) + logP_2(v_i|\pi_t,s) - logQ_\rho(\pi_t|s,v_i) \tag{27}$$

Where $\omega(\pi_t, s, v_i)$ serve as a learning signal in the policy gradient. To reduce the variance of the gradient, we normalize the signal $\omega(\pi_t, s, v_i)$ and subtract the baseline function $b(s, v_i)$ [48], so the gradient can be approximately transformed into:

$$\nabla_\rho L \approx \frac{1}{K}\sum_{j=1}^K \nabla_\rho logQ_\rho(\pi_t|s,v_i)\Big(\frac{\omega(\pi_t,s,v_i) - \tilde{\mu}}{\tilde{\sigma}} - b(s,v_i)\Big) \tag{28}$$

Where $\tilde{\sigma}$ and $\tilde{\mu}$ estimate standard deviation and the mean of $\omega(\pi_t, s, v_i)$ with moving average.

Testing and reasoning In the process of reasoning and testing, it would be too expensive to find the target node sequence only through the problem instance and the following equation:

$$argmax logP_{\theta_2}(v_i|\pi_t,s)P_{\theta_2}(v_i|\pi_t,s) \tag{29}$$

Therefore, we use the method of beam search to approximate the solution. We select k candidate nodes from $P_{\theta_1}(\pi_t|s)$ based on the score (when $k = 1$, it is a greedy algorithm), and the final correct node is:

$$v^* = argmax logP_{\theta_2}(v_i|\pi_t,s), v_i \in G_{\pi_t},\ \pi_t \in \{\pi_1,\pi_2,\ldots,\pi_k\} \tag{30}$$

Algorithm 1 Training of VARL

Initialize θ_1, θ_2, ρ with a small labeled data set
i←0
while not converged, **do**
Sample (s, π_t) from the data distribution D_{train}
Sample v_i using $Q_\rho(\pi_t|s,v_i) \propto \exp(\tilde{W}_{\pi_t}^T \tilde{f}(s) + \tilde{f}_G(s)^T g(G_{v_i \to \pi_t}))$
Sample ϵ from $N(0,1)$
Compute θ_1 - gradient:
$\nabla_{\theta_1} L \leftarrow \frac{1}{K}\sum_{j=1}^K \nabla_{\theta_1} logP_{\theta_1}(s|\pi_{t\rho}(s,\epsilon))$
Compute θ_2 - gradient:
$\nabla_{\theta_2} L \leftarrow \frac{1}{K}\sum_{j=1}^K \nabla_{\theta_2}[-T^*(s,\pi_{t\rho}(s,\epsilon)) + logP_{\theta_2}(s|\pi_{t\rho}(s,\epsilon))]$
Compute ρ - gradient:
$\nabla_\rho L \approx \frac{1}{K}\sum_{j=1}^K \nabla_\rho logQ_\rho(\pi_t|s,v_i)\big(\frac{\omega(\pi_t,s,v_i) - \tilde{\mu}}{\tilde{\sigma}} - b(s,v_i)\big)$
Smoothing $\tilde{\mu}$, $\tilde{\sigma}$ with $\omega(\pi_t,s,v_i)$
Update the baseline $b(a, q)$ using least square
$\rho \leftarrow \rho - \eta\nabla_\rho L$
$\theta_1 \leftarrow \theta_1 - \eta\nabla_{\theta_1} L$
$\theta_2 \leftarrow \theta_2 - \eta\nabla_{\theta_2} L$
i←i + 1
end while

4 Experiments

This section demonstrates results and the discussion of the testing and reasoning and performs comparative analysis to prove the proposed method's effectiveness and efficiency. We first introduced the data set used in the experiment and the parameter settings in the method in Section 4.1. Then we show the comparison of the effect of our method and the baseline on the TSP instance and the VRP instance in Section 4.2 and Section 4.3, respectively. Finally, in Section 4.4, we show the learning curve of our method during training and testing.

4.1 Data sets and Settings

As mentioned above, VARL can solve both TSP and (simple) VRP, so our experiment is conducted in both problem instances. We employ generated data sets similar to previous methods [28]. The input form of the problem example would be: "What is the shortest path for the vehicle to leave from the central node and pass through each customer node?", "What is the shortest path set of N vehicles starting from the central node passing through each customer node and returning to the central node," etc. Generally, the form of problem instances of VRP is relatively fixed, so we only need simple semantic matching to identify the topic of problems. So formally, our framework is relatively more flexible than the previous one because it is problem-oriented rather than designing heuristics based on the problem manually.

In the training phase of algorithm 1, we set the batch size at 512, epoch at 100, and training steps as 2000. We apply the Adam optimizer to update parameters and set the learning rate and learning rate delay at 10^{-3} and 0.96. The previous learning-based methods are primarily based on attention mechanism and GNN, quite different from our model. We choose some recent representative works as baselines, and their experimental results are obtained from their papers. Besides, we choose some heuristics-based methods [28], such as LKH3, Concorde, OR-Tools, and Nearest Neighbor.

4.2 Experiments on TSP

TSP is a simple variant of VRP. In VARL, VRP is transformed into TSP by subgraph decomposition, enabling it to simultaneously solve two kinds of problems. In addition, it can decompose large graphs into small graphs, which is more conducive to solving large-scale instances.

We employ an evaluation method similar to [5] to evaluate solution quality in test cases. We employ the approximate ratio of various methods to the optimal solution, averaged over the set of test cases. The approximation ratio of solution π_t to problem instance s is denoted as $R_a(\pi_t, s) = Max(\frac{OPT(s)}{c(h(\pi_t))}, \frac{c(h(\pi_t))}{OPT(s)})$, where $OPT(s)$ is the optimal solution obtained by the solver and $c(h(\pi_t))$ is the objective value of solution π_t. We select some state-of-the-art learning-based methods (such as AM [28], S2V-DQN [5], and PN-AC [26]) and traditional heuristic algorithms [5] (such as 2-opt, Cheapest, and Nearest) as baselines. From the experimental results in Fig. 3, we can see that the approximate ratio of VARL on the small TSP is lower than that of baselines.

Compared to traditional methods, the 2-opt algorithm based on the improvement heuristic is far higher in solution quality than the simple interpolation algorithms (Cheapest and Nearest) based on the construction heuristic. The latter are also less competitive in the overall comparison. The working principle of the improvement heuristic is to improve the quality of the solution by giving an initial solution and then continuously improving iteratively. The construction heuristic aims to generate a complete solution from scratch. Generally speaking, the quality of the final solution obtained by the improvement heuristic algorithm is higher than that of the construction heuristic algorithm. Still, the improvement heuristic algorithm often depends on the quality of the initial solution, so its generalization is weaker than that of the construction heuristic. Our method and the selected learning-based baseline are essentially learning construction heuristics. However, from the results, we can see that the quality of the solution of VRL surpasses the traditional improvement heuristic algorithm 2-opt and other learning-based methods, which shows that our method effectively learns the construction heuristic and the data distribution in the TSP instance. Compared with multi-stage learning, VARL uses an end-to-end training paradigm to avoid errors caused by cascading propagation, which has a positive effect on improving the accuracy of the solution. Besides, compared to the encoder-decoder framework constructed by applying attention and GNN, VARL seems to be inherently suitable for processing hidden variables in the intermediate process, which is more conducive to acceptable processing variables in the modeling.

From Fig. 3, we can also see that in large-scale TSP instances, VARL still has an advantage in the quality of the solution. Reasoning on large-scale TSP instances is challenging because the complexity of the problem increases geometrically as the number of nodes increases. Traditional methods require a lot of trial and error and expertise and cannot generalize, so designing a targeted heuristic algorithm is expensive. The RNN-based and attention-based approaches show significant performance degradation when the problem size increases due to their inherent mechanisms. For processing large-scale graphs, we can use our proposed GRN and graph decoder to decompose it into sub-graphs based on the identification of the central node and then

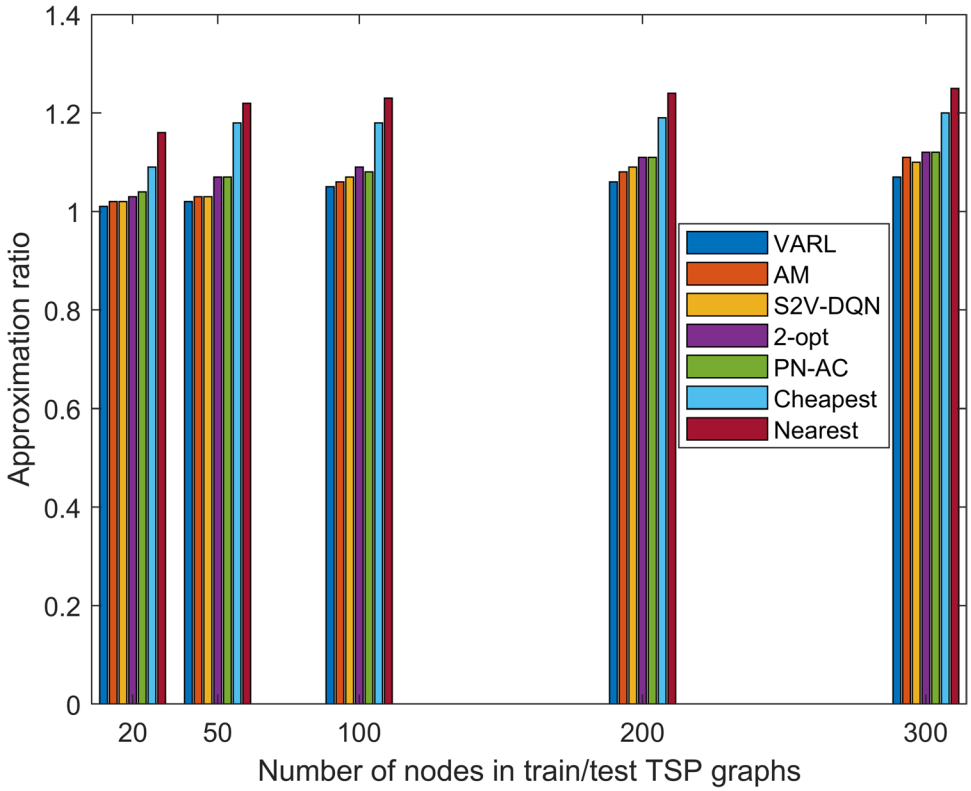

Fig. 3 The approximate ratio of the VARL and baseline methods results in the optimal solution on small TSP instances

divide and process each sub-graph. It is much easier than dealing with large-scale graphs directly. The advantage of VARL is still in both the modeling and training methods, the problem-specific encoding-decoding structure we designed, the GNN processing and reasoning of the whole graph, GAN and REINFORCE with variance reduction optimize the whole framework end-to-end efficiently.

4.3 Experiments on VRP

We solve the more complex constrained path optimization problem (CVRP) by following [27] to generate VRP instances with nodes of 20, 50, 100, respectively, and normalize the demand of each node depending on the capacity. We assume that node locations and requirements are randomly generated from a fixed distribution. The warehouse and customers' locations are randomly generated in a unit square $[0,1] \times [0,1]$. For simplicity, let us assume that the requirements for each customer point are from $\{1, \ldots, 9\}$, while we find any distribution, including continuous distribution, can generate that customer demand. In each decoding step, the vehicle selects the node to visit in the next step from the graph. After visiting the customer node v_i, the demand and vehicle capacity is updated as follows:

$$x^d_{i,t+1} = \max(0, x^d_{i,t} - Q_t) \tag{31}$$

$$x^d_{j,t+1} = x^d_{j,t} \tag{32}$$

$$Q_{t+1} = \max(0, Q_t - x^d_{i,t}) \tag{33}$$

The above variables are defined in Section 3.1. Because we use a messaging passing network (the information for nodes and edges is updated iteratively over time) and a problem instance-oriented mechanism. We can deal with dynamic cases that change as time step t changes.

As shown in Table 1, VARL still has an advantage over traditional and attention-based approaches addressing VRP. VARL is more flexible and needs to change the corresponding problem inputs and related variables to deal with more complex VRP problems. Compared with the method based solely on attention, VARL can better use the advantages of GNN. It has a global and local view on the graph and can better guide the agent to reason. In addition, the REINFORCE with variance reduction in VARL is more efficient than the REINFORCE with rollout baseline in AM, which avoids a lot of Monte Carlo rollouts, and training the autoencoder is significantly lighter than training the transformer.

It is worth noting that the solution solver still maintains a high level in terms of its quality, especially the

Lin-Kernighan-Helsgaun heuristic. However, solution solvers like LKH3 [49] [50] are often based on improvement heuristics, that is, to improve the quality of the solution by continuously improving the initial solution. It is unfair to compare the improvement heuristic with the construction heuristic of learning to construct a solution from scratch. The former has already obtained at least one feasible solution at the beginning of the test. In addition, the time consumed by traditional heuristic algorithms on small-scale problem instances is already too long, which is unacceptable for large-scale problems in reality. For the time complexity of traditional algorithms, there is not much room for improvement. However, as long as the learning-based method is well trained, after learning the data distribution of a given problem, the time it takes to test is much lower than traditional algorithms. More importantly, learning-based methods are much higher than traditional algorithms in terms of generalization because the former can solve problems one-to-many. In contrast, the latter can only solve problems one-to-one.

4.4 Effects of learning

Training effectiveness of learning-based methods is a significant evaluation index, especially for reinforcement learning methods, because different reinforcement learning algorithms significantly impact training effectiveness. For example, Q-learning is more suitable for a greedy algorithm to deal with discrete actions, while REINFORCE is more suitable for continuous actions. The training effect of the reinforcement learning algorithm and space's size will be different due to different models or application scenarios. VARL also combines different components, such as variational autoencoder, GAN, GNN. It is necessary to test the learning effect of VARL, so we observe the comparison between VARL and baselines in terms of convergence on VRP100. Figure 4 visualizes the experimental results.

Table.1 Comparison of model performance tested on different VRP variants with different node numbers

Method	n=20	n=50	n=100
	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 -
Gurobi	**6.10** -	- -	- -
PRL (greedy)	6.59 **-**	11.39 **-**	17.23 **-**
PRL (beam)	6.40 **-**	11.15 -	16.96 **-**
AM (greedy)	6.40 **1 s**	10.98 3 s	16.80 8 s
AM (sampling)	6.25 6 m	10.62 28 m	16.23 2 h
VARL	6.21 **1 s**	10.54 **2 s**	16.19 **5 s**

From the learning curve, we can see that the convergence speed and learning robustness of VARL have advantages over the baseline. Compared with GCN-NPEC [12], its two-stage training strategy can make the model a "warm start" initially. Still, the training loss will be back-propagation, and the subsequent reinforcement learning will lead to more and more errors, and its training method is also the REINFORCE with rollout baseline. VARL uses the end-to-end training method, and GAN is introduced to assist optimization. Its training method is more efficient and stable. From the perspective of the whole framework, VARL is still much lighter than GCN-NPEC because GCN-NPEC contains more GNN, so it is a heavy burden for the joint training of multi-GNN.

We demonstrate in Fig. 5 the learning curve of using reinforcement learning to train VRL on TSP20. The curve describes the change in the agent's rewards with increasing time steps in each epoch. We selected the first five epochs because we observed that the learning curve has stabilized only after the first epoch, and the learning curve has converged at the beginning of the first epoch. It shows that our method can quickly converge on small-scale problem instances and effectively learn the data distribution. Reinforcement learning is notoriously unstable, but we can see that REINFORCE with variance reduction in VARL is efficient from the learning curve.

5 Conclusions

In this paper, we apply variational learning to combinatorial optimization (CO) on graphs. Firstly, we define the combinatorial optimization problem in the form of variational learning. We use the message passing network to process the input graph. At the same time, we combine the node information in the graph to identify the central node (topic) in the problem instance and decompose the graph into different sub-graphs with the central node as the root. We propose a novel GNN (Graph Reasoning Network) for reasoning on graphs and introduce GAN in variational learning to improve efficiency and robustness. Finally, we use REINFORCE with variance reduction to train the entire framework. We introduce variational learning into CO for the first time and prove that the fusion of each component is appropriate.

Variational autoencoders are inherently equipped with encoders and decoders, in line with the paradigm of solving CO. The latent variables in variational learning can enable the model to learn better data distribution. More importantly, we can employ variational learning as a basic framework to incorporate different efficient learning algorithms. In the future, we will continue to explore the application and

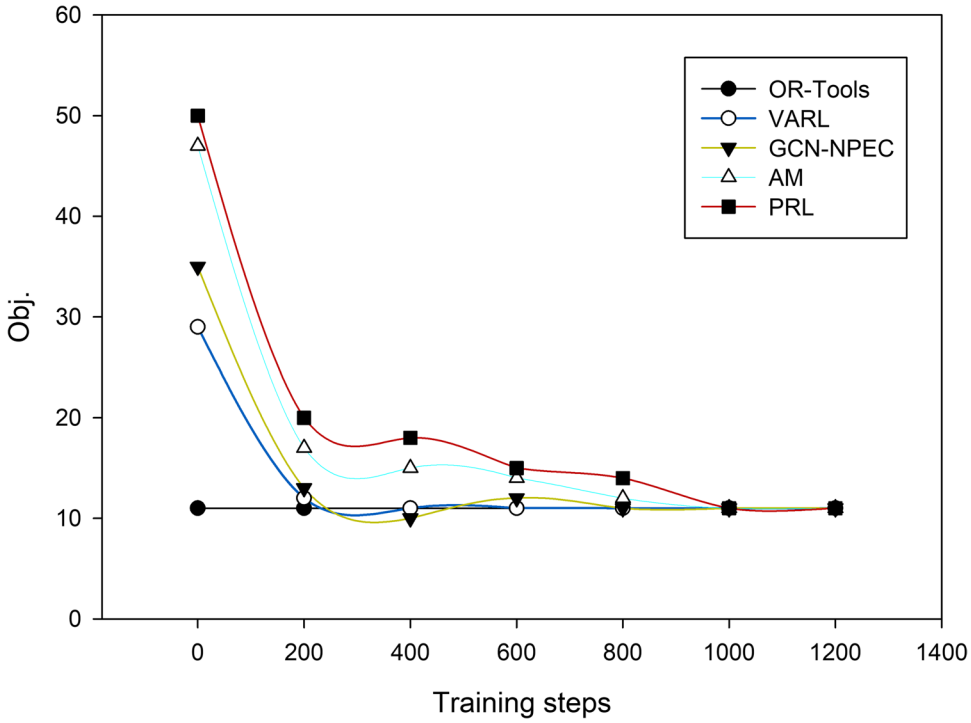

Fig. 4 The approximate ratio of the VARL and baseline methods results in the optimal solution on VRP100

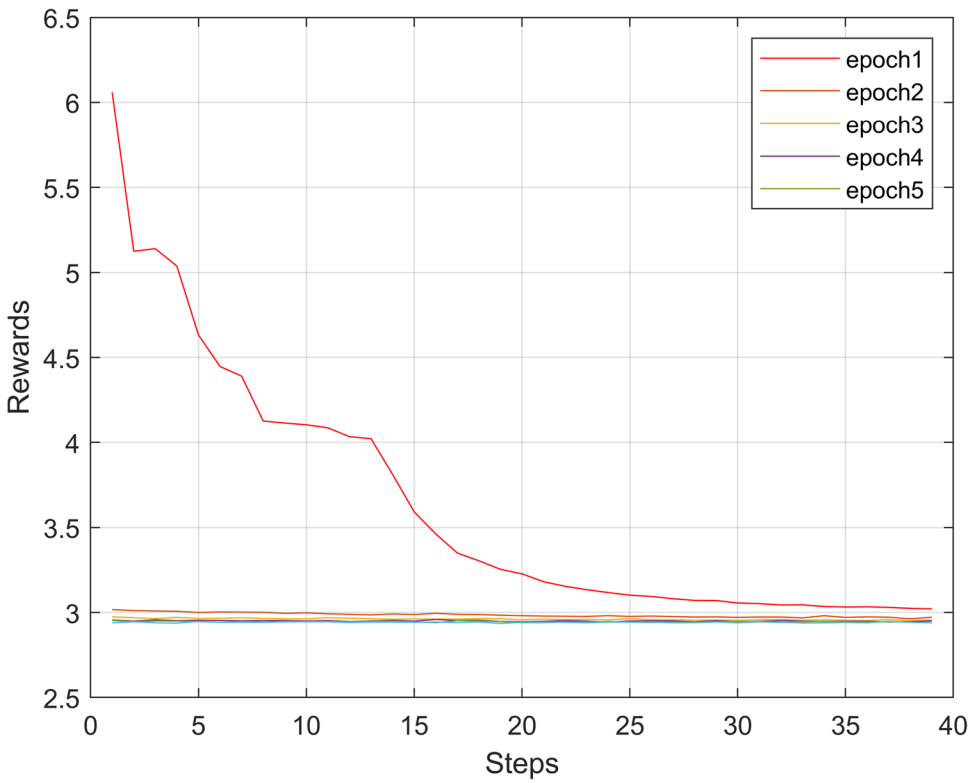

Fig. 5 We selected the first five epochs of VARL training on TSP20 and generated a curve of the average route length (reward value) obtained with each epoch's increment of the time step

innovation of probabilistic graph model modeling and its training method of CO on graphs.

Acknowledgements The author thanks Chunlei Tang, Ph.D., Yun Xiong, Ph.D., and Yangyong Zhu, Ph.D., for valuable comments on the early versions.

References

1. Goyal S (2010) A survey on travelling salesman problem. Midwest Instr. Comput. Symp. 1–9
2. Alba E, Dorronsoro B (2008) Logistics: the vehicle routing problem, 175–186 https://doi.org/10.1007/978-0-387-77610-1_13
3. Hsieh FS, Guo YH (2019) A discrete cooperatively coevolving particle swarm optimization algorithm for combinatorial double auctions. Appl Intell 49:3845–3863. https://doi.org/10.1007/s10489-019-01556-8
4. Zhang W, Gao K, Zhang W, Wang X, Zhang Q, Wang H (2019) A hybrid clonal selection algorithm with modified combinatorial recombination and success-history based adaptive mutation for numerical optimization. Appl Intell 49:819–836. https://doi.org/10.1007/s10489-018-1291-2
5. Dai H, Khalil EB, Zhang Y, Dilkina B, Song L (2017) Learning combinatorial optimization algorithms over graphs. Adv Neural Inf Process Syst :6349–6359
6. Jordan MI, Mitchell TM (2015) Machine learning: Trends, perspectives, and prospects. Nature 349
7. Lecun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521:436–444. https://doi.org/10.1038/nature14539
8. Mousavi SS, Schukat M, Howley E (2018) Deep reinforcement learning: an overview. Lect Notes Netw Syst 16:426–440. https://doi.org/10.1007/978-3-319-56991-8_32
9. Wang Q, Tang C (2021) Deep reinforcement learning for transportation network combinatorial optimization: A survey. Knowl Based Syst 233:107526. https://doi.org/10.1016/j.knosys.2021.107526
10. Vinyals O, Babuschkin I, Czarnecki WM, Mathieu M, Dudzik A, Chung J, Choi DH, Powell R, Ewalds T, Georgiev P, Oh J, Horgan D, Kroiss M, Danihelka I, Huang A, Sifre L, Cai T, Agapiou JP, Jaderberg M, Vezhnevets AS, Leblond R, Pohlen T, Dalibard V, Budden D, Sulsky Y, Molloy J, Paine TL, Gulcehre C, Wang Z, Pfaff T, Wu Y, Ring R, Yogatama D, Wünsch D, McKinney K, Smith O, Schaul T, Lillicrap T, Kavukcuoglu K, Hassabis D, Apps C, Silver D (2019) Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature 575:350–354. https://doi.org/10.1038/s41586-019-1724-z
11. Ecoffet A, Huizinga J, Lehman J, Stanley KO, Clune J (2021) First return, then explore. Nature 590:580–586. https://doi.org/10.1038/s41586-020-03157-9
12. Duan L, Zhan Y, Hu H, Gong Y, Wei J, Zhang X, Xu Y (2020) Efficiently solving the practical vehicle routing problem: a novel joint learning approach. Proc ACM SIGKDD Int Conf Knowl Discov Data Min :3054–3063. https://doi.org/10.1145/3394486.3403356
13. Manchanda S, Mittal A, Dhawan A, Medya S, Ranu S, Singh A (2019) Learning Heuristics over Large Graphs via Deep Reinforcement Learning. http://arxiv.org/abs/1903.03332
14. Kingma DP, Welling M (2014) Auto-encoding variational bayes. 2nd Int. Conf. Learn. Represent. ICLR 2014 - Conf Track Proc 1–14
15. Zhu D, Wang D, Cui P, Zhu W (2018) Deep variational network embedding in wasserstein space. Proc ACM SIGKDD Int Conf Knowl Discov Data Min 2827–2836. https://doi.org/10.1145/3219819.3220052
16. Bengio Y, Lodi A, Prouvost A (2021) Machine learning for combinatorial optimization: A methodological tour d'horizon. Eur J Oper Res 290:405–421. https://doi.org/10.1016/j.ejor.2020.07.063
17. Rumelhart DE, Hinton GE, Williams RJ (1986) Learning representations by back-propagating errors. Nature 323:533–536. https://doi.org/10.1038/323533a0
18. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I (2017) Attention is all you need. In: Advances in Neural Information Processing Systems, pp 5999–6009
19. Xu K, Jegelka S, Hu W, Leskovec J (2019) How powerful are graph neural networks? 7th Int. Conf. Learn. Represent. ICLR 2019
20. Mazyavkina N, Sviridov S, Ivanov S, Burnaev E (2021) Reinforcement learning for combinatorial optimization: A survey. Comput Oper Res 134:0–2. https://doi.org/10.1016/j.cor.2021.105400
21. Hopfield JJ, Tank DW (1985) "Neural" computation of decisions in optimization problems. Biol Cybern 52:141–152. https://doi.org/10.1007/BF00339943
22. Sutskever I, Vinyals O, Le QV (2014) Sequence to sequence learning with neural networks. Adv Neural Inf Process Syst 4:3104–3112
23. Vinyals O, Fortunato M, Jaitly N (2015) Pointer networks. Adv Neural Inf Process Syst, 2692–2700
24. Ivanov S, D'yakonov A (2019) Modern deep reinforcement learning algorithms. arXiv.
25. Bahdanau D, Brakel P, Xu K, Goyal A, Courville A, Pineau RLJ, Bengio Y (2017) An actor-critic algorithm for sequence prediction. 5th Int Conf Learn Represent ICLR 2017 - Conf Track Proc, 1–17
26. Bello I, Pham H, Le QV, Norouzi M, Bengio S (2017) Neural combinatorial optimization with reinforcement learning. 5th Int. Conf. Learn. Represent. ICLR 2017 - Work. Track Proc, 1–15
27. Nazari M, Oroojlooy A, Takáč M, Snyder LV (2018) Reinforcement learning for solving the vehicle routing problem. Adv Neural Inf Process Syst, 9839–9849
28. Kool W, Van Hoof H, Welling M (2019) Attention, learn to solve routing problems! 7th Int. Conf. Learn. Represent. ICLR 2019. 1–25
29. Veličković P, Casanova A, Liò P, Cucurull G, Romero A, Bengio Y (2018) Graph attention networks. 6th Int. Conf. Learn. Represent. ICLR 2018 - Conf. Track Proc, 1–12
30. Defferrard M, Bresson X, Vandergheynst P (2016) Convolutional neural networks on graphs with fast localized spectral filtering. Adv Neural Inf Process Syst, 3844–3852
31. Li Z, Chen Q, Koltun V (2018) Combinatorial optimization with graph convolutional networks and guided tree search. Adv Neural Inf Process Syst, 539–548
32. Ma Q, Ge S, He D, Thaker D, Drori I (2019) Combinatorial optimization by Graph Pointer Networks and Hierarchical Reinforcement Learning. arXiv
33. Cappart Q, Goutierre E, Bergman D, Rousseau L-M (2019) Improving optimization bounds using machine learning: decision diagrams meet deep reinforcement learning. Proc AAAI Conf Artif Intell 33:1443–1451. https://doi.org/10.1609/aaai.v33i01.33011443
34. Yolcu E, Póczos B (2019) Learning local search heuristics for Boolean satisfiability. NeurIPS, 7992–8003
35. Barrett T, Clements W, Foerster J, Lvovsky A (2020) Exploratory combinatorial optimization with reinforcement learning. https://doi.org/10.1609/aaai.v34i04.5723
36. Beloborodov D, Ulanov AE, Foerster JN, Whiteson S, Lvovsky AI (2021) Reinforcement learning enhanced quantum-inspired

algorithm for combinatorial optimization. Mach Learn Sci Technol 2:025009. https://doi.org/10.1088/2632-2153/abc328
37. Chen X, Tian Y (2019) Learning to perform local rewriting for combinatorial optimization. Adv Neural Inf Process Syst 32
38. Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, Hubert T, Baker L, Lai M, Bolton A, Chen Y, Lillicrap T, Hui F, Sifre L, Van Den Driessche G, Graepel T, Hassabis D (2017) Mastering the game of Go without human knowledge. Nature 550:354–359. https://doi.org/10.1038/nature24270
39. Schrittwieser J, Antonoglou I, Hubert T, Simonyan K, Sifre L, Schmitt S, Guez A, Lockhart E, Hassabis D, Graepel T, Lillicrap T, Silver D (2020) Mastering Atari, Go, chess and shogi by planning with a learned model. Nature 588:604–609. https://doi.org/10.1038/s41586-020-03051-4
40. Huang J, Patwary M, Diamos G (2019) Coloring big graphs with AlphaGoZero. arXiv
41. Wang Q, Hao Y, Cao J (2021) Learning to traverse over graphs with a Monte Carlo tree search-based self-play framework. Eng Appl Artif Intell 105:104422. https://doi.org/10.1016/j.engappai.2021.104422.
42. Laterre A, Fu Y, Jabri MK, Cohen A-S, Kas D, Hajjar K, Dahl TS, Kerkeni A, Beguir K (2018) Ranked reward: enabling self-play reinforcement learning for combinatorial optimization. arXiv
43. Mansimov E, Parisotto E, Ba JL, Salakhutdinov R (2016) Generating images from captions with attention. 4th Int. Conf. Learn. Represent. ICLR 2016 - Conf. Track Proc, 1–12
44. Guu K, Hashimoto TB, Oren Y, Liang P (2017) Generating sentences by editing prototypes. arXiv 2. https://doi.org/10.1162/tacl_a_00030
45. Mahdavi S, Khoshraftar S, An A (2020) Dynamic joint variational graph autoencoders. Commun Comput Inf Sci 1167 CCIS:385–401. https://doi.org/10.1007/978-3-030-43823-4_32
46. Gilmer J, Schoenholz SS, Riley PF, Vinyals O, Dahl GE (2017) Neural message passing for quantum chemistry. 34th Int. Conf. Mach. Learn. ICML 3:2053–2070
47. Willia RJ (1992) Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach Learn 8:229–256. https://doi.org/10.1023/A:1022672621406
48. Mnih A, Gregor K (2014) Neural variational inference and learning in belief networks. 31st Int. Conf. Mach. Learn. ICML 5:3800–3809
49. Zheng J, He K, Zhou J, Jin Y, Li C.-M (2020) Combining reinforcement learning with Lin-Kernighan-Helsgaun algorithm for the traveling salesman problem. Assoc Adv Artif Intell
50. Helsgaun K (2009) General k-opt submoves for the Lin-Kernighan TSP heuristic. Math Program Comput 1:119–163. https://doi.org/10.1007/s12532-009-0004-6

Qi Wang received a B.S. and M.Eng. degree in software engineering from Jilin University (Changchun city) and Central South University (Changsha city) in 2012 and 2016, China, respectively. He is currently pursuing a Ph.D. degree in software engineering at the school of computer science, Fudan University, Shanghai, China.

His current research interests include combinatorial optimization, deep learning, and reinforcement learning.