Multi-objective Adaptive Dynamics Attention Model to Solve Multi-objective Vehicle Routing Problem

Guang Luo 2110436129@email.szu.edu.cn
Jianping Luo * ljp@szu.edu.cn
Guangdong Key Laboratory of Intelligent Information Processing, College of Electronic and Information Engineering, Shenzhen University, China

Editors: Berrin Yanıkoğlu and Wray Buntine

Abstract

Multi-objective combinatorial optimization problems (MOCOP) are commonly encountered in everyday life. However, finding the optimal solution through traditional exact and heuristic algorithms can be time-consuming due to its NP-hard nature. Fortunately, deep reinforcement learning (DRL) has shown promise in solving complex combinatorial optimization problems (COP). In this paper, we introduce a new Multi-objective Adaptive Dynamics Attention Model (MOADAM) that aims to better approximate the whole Pareto set. We modify the encoder and decoder of the model to better utilize dynamic information, and we also design a new weight sampling method to improve the model's performance for extreme solutions. Our experimental results demonstrate that our proposed model outperforms the current state-of-the-art algorithm in terms of solution quality on multi-objective vehicle routing problems with capacity constraints (MOCVRP).

Keywords: Vehicle routing problems with capacity constraints (CVRP), Multi-objective combinatorial optimization problems (MOCOP), Deep reinforcement learning (DRL), Attention mechanism, Weight.

1. Introduction

MOCVRP is a combination of two different problems, namely CVRP and MOCOP. In CVRP, there is a set of goods that need to be transported from the depot to various customers, each with a specific demand. The transportation is carried out using a certain number of vehicles with a predetermined capacity limit. The objective of the problem is to determine the optimal set of routes that can deliver all the goods required by the customers. Each customer can only be visited once, and the vehicle must not exceed its capacity limit during transportation, except for the depot. For MOCOP, a general definition is as follows:

$$\begin{aligned} & f(x) = (f_1(x), f_2(x), \ldots, f_m(x)) \\ & s.t. \quad x \in X \end{aligned} \tag{1}$$

MOCOP consists of different objective functions and a decision space $X \in \mathbb{R}^n$. As these objectives often conflict, improving one objective may lead to a decrease in the results of other objectives. Therefore, they must be coordinated and compromised to achieve the best possible values for each sub-objective. This is known as the Pareto optimal solution. For

* Corresponding author

two solutions $u, v \in \mathbb{R}^m$, if and only if $u_i \leq v_i$ for $\forall i \in \{1, 2, \ldots, m\}$ and there exists at least one $j \in \{1, 2, \ldots, m\}$ such that $u_i < v_i$, then u is said to dominate v. If a solution is not dominated by any other solution in the Pareto set (PS), it is known as a Pareto optimal solution $x^* \in X$. All Pareto optimal solutions form a Pareto set.

Decomposition is the mainstream strategy for solving MOCOP. It divides the multi-objective problem into multiple sub-problems. Zhang and Li (2007) and its variants Trivedi et al. (2016) collaboratively solve these sub-problems and generate a limited Pareto set to approximate the Pareto front. The most commonly used method for constructing single-objective sub-problems is weight-based scalarization (Ehrgott (2005), Miettinen (2012)).

For an m-objective optimization problem, the weight vector of the objective function can be defined as $\lambda_i \in R^m$, subject to the constraints $\lambda_i \geq 0$ and $\sum_{i=1}^{m} \lambda_i = 1$. Multi-objective optimization decomposition methods are generally divided into weighted sum aggregation and weighted Chebyshev aggregation.

Weighted sum aggregation is the simplest method, which defines the minimized aggregation function associated with λ as follows:

$$f^{ws}(x \mid \lambda) = \sum_{i=1}^{m} \lambda_i f_i(x) \tag{2}$$

Weighted Chebyshev aggregation is another method, defined as follows:

$$f^{ws}(x \mid \lambda) = \max_{i=1}^{m} \{\lambda_i \mid f_i(x) - z_i^* \mid\} \tag{3}$$

Here, $z_i^* = \min_{x \in X} f_i(x)$ represents the ideal value of $f_i(x)$.

We are faced with the question of whether the weights are known or not for MOCOP. Training a model with specific weights is not practical as it can only solve specific sub-problems that have predetermined weights for each objective. Therefore, we consider the scenario where the weights are not known in advance. We can generate a Pareto set based on different weights using the decomposition method.

In recent years, researchers have proposed many methods to solve MOCVRP, such as NSGA-II Deb et al. (2002) and MOEA/D Zhang and Li (2007). However, these methods usually require carefully crafted and specialized heuristics for each problem, which can be very labor-intensive in practice. At the same time, these evolutionary algorithms require iterative solutions for each different instance, and once there is a new instance, they need to be resolved, which lacks timeliness, such as Pareto Local Search (PLS) Angel et al. (2004) and Multi-Objective Genetic Local Search Algorithm (MOGLS) Jaszkiewicz (2002).

To effectively solve MOCVRP, machine learning technology is a promising approach as it helps to learn patterns behind problem instances. DRL is a rapidly growing technology that utilizes neural networks to approximate solutions to MOCOP. A recent study by Lin et al. (2021) uses a hypernetwork to generate different decoder parameters for different objective weight vectors, which inspired our research. This paper introduces a new attention-based DRL model called MOADAM that is designed to efficiently obtain the Pareto set for MOCVRP using a single model. The main objective of this research is to create a universal model that can effectively utilize existing information, rather than increasing model complexity. Our experiments have shown that incorporating such information as input can

significantly improve the model's learning ability. To address the MOCVRP challenge, we break down the problem into two parts: CVRP and MOCOP. We first improve the performance of CVRP before expanding the model to tackle MOCVRP for further optimization. The key contributions of this paper can be summarized as follows:

1) In terms of CVRP optimization, we propose a new model which can effectively aggregate critical information. Our approach involves reordering the encoder component and incorporating new modules to optimize edge information utilization. Additionally, we have enriched the context embedding of the decoder component by integrating dynamic information to enhance the decoding process.

2) In terms of MOCOP optimization, we first introduced a novel approach of incorporating weight information through a simple MLP to generate weight embeddings. By combining these weight embeddings with node embeddings, they serve as input to the encoder component, allowing for the encoder to adapt to different weight combinations. Furthermore, we have also designed a new weight sampling method, which enhances the model's ability to learn more extreme solutions. Both improvements can improve the convergence of the Pareto set.

3) Experimental results show that our proposed model outperforms the current state-of-the-art DRL learning-based methods in MOCVRP, exhibiting better solution quality.

2. Related work

MOCVRP is a combination of both CVRP and MOCOP, and we first focus on CVRP before expanding to MOCVRP. In the past few decades, numerous solvers have been proposed for CVRP, which can be broadly classified into two categories: exact solvers and approximate solvers.

Exact solvers: Exact solvers, such as the algorithm proposed by Desrochers et al. (1992) based on branch and bound and pricing strategies for solving solve vehicle routing problems (VRP). There are also some exact algorithms for solving VRP, such as Ropke and Pisinger (2006), Subramanian and Ukkusuri (2014), Focacci et al. (2007). Although these exact solvers can provide optimal solutions, they have long running times and are difficult to handle large-scale problems. In addition, these algorithms typically require higher expertise, making them difficult for general users to use.

Approximate solvers: Approximate solvers can be divided into two categories: those based on heuristic search algorithms and those based on end-to-end solvers using DRL.

For the former, a complete set of solutions is first initialized, and in each iteration, this set is modified using search operators to obtain another complete set of solutions. This process continues until an approximate solution to the global optimal solution is achieved. Heuristic search-based solvers for CVRP can be classified into several types, such as the greedy algorithm, genetic algorithm, simulated annealing algorithm, taboo search algorithm, simulated particle swarm, ant algorithm, and local search. Scholars have also found that combining multiple algorithms can complement each other's strengths. CVRP can be transformed into MOCVRP by using a fitness function, and heuristic solvers can generate a Pareto set through non-domination sorting or crowding distance comparison. Classic solvers such as NSGA-II Deb et al. (2002), MOEA/D Zhang and Li (2007). Wang et al. (2015) used the multi-objective local search and multi-objective memetic algorithm to solve simultaneous

pickup and delivery and time windows. Although using these solvers can obtain a Pareto set, they lack flexibility and need to be reinitialized when encountering new problem instances.

For the latter, we first train a neural network model, which takes the problem instance as input and outputs a sequence of decisions, allowing for a direct solution to the problem instance. Compared to the solvers mentioned above, this solver has greater flexibility because it can quickly generate corresponding solutions for different problem instances. Hopfield and Tank (1985) first used the Hopfield neural network model to solve optimization problems. With the advancement of DRL technology, recent work has sought to design solvers based on deep neural networks. Sutskever et al. (2014) proposed a neural network-based sequence-to-sequence learning method, also known as an encoder-decoder model. Vinyals et al. (2015) applied this method and proposed a pointer network. Bello et al. (2017) first used the DRL method to solve TSP and used the pointer network framework. Nazari et al. (2018) pointed out the limitations of the pointer network and replaced it with a DRL algorithm based on a Monte Carlo tree search. Kool et al. (2019) proposed using the transformer architecture based on the attention model (AM) Vaswani et al. (2017) and introduced a greedy rollout baseline estimator for ease of training. Due to its flexibility, most subsequent work is based on it. We plan to follow the AM to design a flexible universal framework to support the solution of different types of CVRP. Expanding VRP to MOCVRP poses a significant challenge for end-to-end DRL solvers. A specific DRL model can only generate optimal solutions for that particular sub-problem. As far as we know, there is currently limited research on applying DRL to solve MOCVRP. Li et al. (2021) first trains a specifically weighted sub-problem and then generates other weight models through parameter transfer techniques. Zhang et al. (2022) first trains a meta-model, and fine-tunes it to obtain the corresponding weight model for each specific weight sub-problem. Although the above methods can generate a Pareto set, they all require generating different models for each sub-problem to generate solutions. As the number of solutions increases, the training cost will greatly increase.

The recent work by Lin et al. (2021) proposed a novel and advanced method that only needs to train one model to obtain a Pareto set. They used a hypernetwork to change the parameters of W_Q,W_K, and W_V in the decoder, the decoder can be classified accordingly to different weights to obtain different parameters. This method is a new principle method for solving MOCOP and is the most advanced for MOCVRP. Inspired by this method, we have employed a similar method to theirs in tackling MOCVRP and proposed a model called **Multi-objective Adaptive Dynamics Attention Model (MOADAM)**.

3. Preliminary

In this section, we briefly describe some strategies and methods for solving MOCVRP.

3.1. Benchmark Problems

To formalize MOCVRP, we first define a fully connected graph $G = (N, E)$ with n nodes, where $N = \{0, 1, \ldots, n\}$ denote the set of nodes, with 0 representing the depot node and the rest representing customer nodes. $E = \{e_{ij} \mid i, j \in N, i \neq j\}$ denote the set of edges, where e_{ij} denotes the Euclidean distance between node i and node j. We define K as the set of vehicles and Q as the capacity limit of each vehicle. $C = (c_0, \ldots, c_i, \ldots, c_n)$ denote

the set of demands for each node, and c_i denotes the demand of customer node i. A solution can be represented as $\boldsymbol{\pi} = \left(\pi^1, \ldots, \pi^k\right)$, where π^k denotes the route of the k-th vehicle. For each vehicle, the route starts from the depot and ends at the depot. The length of the route formed by the k-th vehicle and the capacity constraints can be calculated as follows:

$$L_k = \sum_{i=0}^{n^k-1} e_{\pi_i^k \pi_{i+1}^k} + e_{\pi_{n^k}^k \pi_0^k}, \quad D_k = \sum_{i=1}^{n^k} c_{\pi_i^k} \leq Q \tag{4}$$

Here, $k \in K$, n^k represents the total number of nodes accessed by the k-th vehicle, and π_i^k represents the i-th node in the route of the k-th vehicle. D_k is the total demand required to be satisfied by the k-th vehicle in its current route.

Similar to Castro-Gutierrez et al. (2011), we choose the total distance $f_1(\pi)$ and the longest route (referred to as the makespan) $f_2(\pi)$ among all routes as our optimization objectives.

$$f_1(\boldsymbol{\pi}) = \sum_{k=1}^{K} L_k, \quad f_2(\boldsymbol{\pi}) = \max\left(L_1, L_2, \ldots L_K\right) \tag{5}$$

Our objective is to:

$$\min f(\pi) = \left(f_1(\boldsymbol{\pi}), f_2(\boldsymbol{\pi})\right) \tag{6}$$

3.2. Decomposition Strategy

In Section 1, we discussed the decomposition strategy for MOCOP. We compared two methods in our experiments and found that using weighted sum aggregation for decomposition could achieve better results. Finally, our reward function is as follows:

$$\begin{aligned} & f^{ws}\left(\pi \mid \lambda^j\right) = \lambda_1^j f_1(\boldsymbol{\pi}) + \lambda_2^j f_2(\boldsymbol{\pi}) \\ & s.t. \quad \lambda_1^j, \lambda_2^j \geq 0, \lambda_1^j + \lambda_2^j = 1 \end{aligned} \tag{7}$$

Where λ^j denotes the specific weight of the j-th sub-problem.

For each sub-problem, we can obtain a solution, and when all sub-problems are solved, we will obtain a Pareto set. However, there is an issue. It is obvious that $f_1(\boldsymbol{\pi})$ is much larger than $f_2(\boldsymbol{\pi})$, and if we do not balance these two objectives, the model will tend to minimize $f_1(\boldsymbol{\pi})$ without considering the weight, resulting in more solutions in the Pareto set being concentrated on one side. This is not the result we want to see. We hope that the solutions on the Pareto set can be more evenly distributed. In the multi-objective reinforcement learning process, we construct n different routes for the same instance. We select the minimum value of $\left\{f_1\left(\pi^1\right), \ldots, f_1\left(\pi^n\right)\right\}$ and $\left\{f_2\left(\pi^1\right), \ldots, f_2\left(\pi^n\right)\right\}$ among all the routes, denoted as $(f_1(\boldsymbol{\pi})^*, f_2(\boldsymbol{\pi})^*)$, to perform normalization for $f^{ws}\left(\boldsymbol{\pi} \mid \lambda^j\right)$. The reward function is modified as follows:

$$f^{ws}\left(\boldsymbol{\pi} \mid \lambda^j\right) = \lambda_1^j f_1(\boldsymbol{\pi}) + \lambda_2^j \frac{f_1(\boldsymbol{\pi})^*}{f_2(\boldsymbol{\pi})^*} f_2(\boldsymbol{\pi}) \tag{8}$$

3.3. Solution construction

The model defines a parameterized by $\boldsymbol{\theta}(\lambda)$ weighted conditional random policy $p_{\boldsymbol{\theta}(\lambda)}$. The aim is to learn an optimal weighted conditional policy $p_{\boldsymbol{\theta}(\lambda)}(\pi \mid s)$ to construct routes with the lowest cost for each weight. The task is formalized as follows: given a problem instance as input, we first encode nodes as node embeddings and keep them unchanged throughout the task. Then, we perform a decoding process for N time steps. For each time step t, the decoder selects an unvisited node, denoted as the node picked at time step t, and represented as π_t. The conditional random policy is as follows:

$$p_{\boldsymbol{\theta}(\lambda)}(\boldsymbol{\pi} \mid s) = \prod_{t=1}^{n} p_{\boldsymbol{\theta}(\lambda)}\left(\boldsymbol{\pi}_t \mid s, \boldsymbol{\pi}_{0:t-1}\right) \tag{9}$$

4. The proposed model: MOADAM

In this section, we mainly describe the structure of the model we propose. MOADAM is based on the work of Lin et al. (2021) and has been improved upon. Following the order of related work introduction in section 2, we will now introduce the improvements made to optimize CVRP.

4.1. Optimization of CVRP

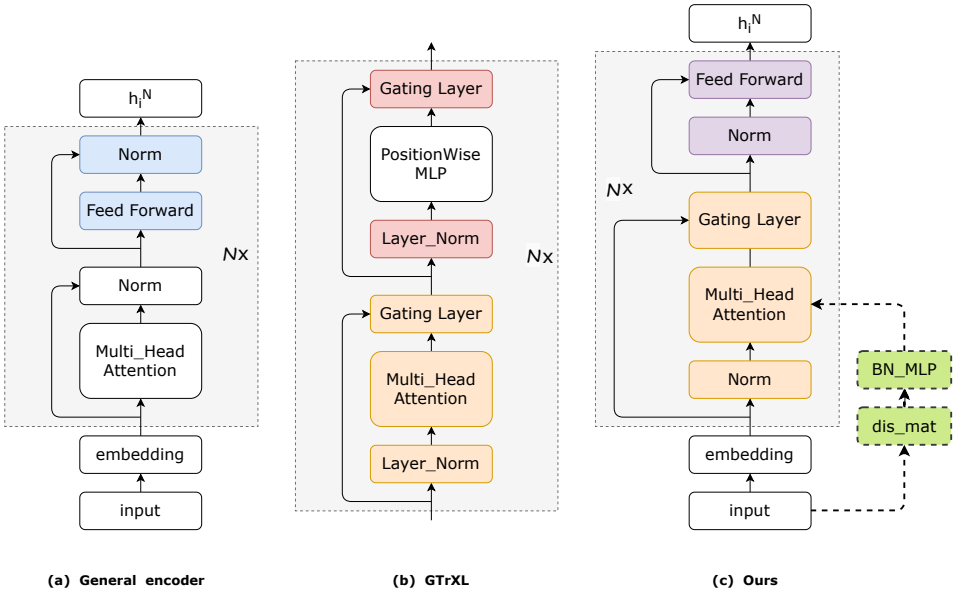

Figure 1: (a) shows the basic framework of the Transformer encoder, (b) presents the GTrXL model proposed in Yu et al. (2018), and (c) illustrates our encoder model. The colored part is our improvement.

Encoder: For each node $i \in \{0, 1, \ldots, n\}$, where 0 represents the depot node, its feature vector is represented as $n_i = [x_i, y_i, d_i]$. Next, the feature vector n_i of each node is mapped to an embedding h_i^0 of dimension d_h through a linear layer.

$$h_i^0 = W_i n_i + b_i \tag{10}$$

Where W_i is trainable parameter matrices, and b_i is trainable bias vectors. Next, $\mathcal{N}$ attention layers further encode the node embeddings $\left\{h_0^0, \ldots, h_n^0\right\}$. Each attention layer consists of a multi-head attention (MHA) sub-layer and a feed-forward (FF) sub-layer. To accelerate deep network training, Kool et al. (2019) used batch normalization (BN) and residual connections in their model. To better utilize information, we have noticed that Yu et al. (2018) proposed a Transformer variant named GTrXL. As shown in Fig. 1, they reorder the normalization process and change the residual connections to be aggregated by gating layers. The gating layer is implemented using a GRU gate. Inspired by this, we modified the model proposed by Kool et al. (2019). Since we do not need to use positional encoding and we found that using only one gating layer in the MHA sub-layer has good learning performance, Therefore, in the FF sub-layer, we reorder the normalization layer and still aggregate it through residual connections. In the MHA sub-layer, we directly integrate edge information into it. First, we obtain a distance matrix between each pair of nodes:

$$E_{\text{dis}} = [e_{00}, e_{01} \cdots, e_{nn}] \tag{11}$$

$$\mathcal{H}_{edge} = W_{e2}\left(BN\left(W_{e1} E_{\text{dis}} + b\right)\right) \tag{12}$$

Where W_{e1} and W_{e2} are trainable parameter matrices, and b is a bias matrix. Therefore, the output of each multi-head attention layer can be calculated as follows:

$$h_i^l = GL\left(h_i^{l-1}, MHA\left(BN\left(H_i^{l-1}\right), \mathcal{H}_{\text{edge}}\right)\right). \tag{13}$$

Here, $l \in \{1, \ldots, \mathcal{N}\}$ represents the l-th sub-attention layer, and $H_i^{l-1} = \left\{h_0^{l-1}, \ldots, h_n^{l-1}\right\}$ represents the set of node embeddings from the $(l-1)$ layer. MHA is based on a self-attention mechanism with M heads, and its calculation can be expressed as follows:

$$q_i = W_{Q_m}^l h_{i_m}^{l-1}, \quad k_i = W_{Q_m}^l h_{i_m}^{l-1}, \quad v_i = W_{V_m}^l h_{i_m}^{l-1} \tag{14}$$

$$u_{ij} = \begin{cases} \dfrac{q_i^T k_j}{\sqrt{d}}, \text{ if } i \text{ adjacent to } j \\ -\infty, \text{ otherwise} \end{cases} \tag{15}$$

$$\epsilon_{ij} = u_{ij} + \mathcal{H}_{ij}, \quad a_{ij} = \frac{e^{\epsilon_{ij}}}{\sum_{k=0}^{n} e^{\epsilon_{ik}}}, \quad \tilde{h}_{i_m}^l = \sum_{j=0}^{n} a_{ij} v_j \tag{16}$$

Here, m represents the m-th head in the multi-head attention mechanism. $W_{Q_m}^l, W_{K_m}^l$, and $W_{V_m}^l$ are trainable parameter matrices. $h_{i_m}^{l-1}$ represents the node embedding for the i-th node in the m-th head at $l-1$ layer. d is defined as $d = d_h/M$. $\mathcal{H}_{ij}$ represents the edge embedding for e_{ij}. We directly aggregate the edge embedding with the self-attention layer

to enable the self-attention layer to modify the attention towards each node based on edge information. Finally, we aggregate the information from all M heads.

$$\tilde{h}_{i_M}^l = \mathrm{MHA}\left(BN\left(H_i^{l-1}\right), \mathcal{H}_{edge}\right) = \sum_{m=1}^{M} W_{o_m}^l \tilde{h}_{i_m}^l \tag{17}$$

Here, $W_{o_m}^l$ represents the trainable parameter matrix for the output projection layer. The gating layer performs aggregation using the following equation:

$$\tilde{h}_i^l = GL\left(h_i^{l-1}, \tilde{h}_{i_M}^l\right) = \left(\mathrm{Sigmoid}\left(W_g h_i^{l-1} + b_g\right) * \tilde{h}_{i_M}^l + h_i^{l-1}\right). \tag{18}$$

After aggregation, we further apply a fully connected layer (FF layer).

$$h_i^l = \tilde{h}_i^l + W_2^l \,\mathrm{ReLU}\left[W_1^l\left(BN\left(\tilde{h}_i^l\right)\right) + b_1^l\right] + b_2^l \tag{19}$$

After $\mathcal{N}$ attention layers, we obtain the final embeddings as $H_i^{\mathcal{N}} = \left\{h_0^{\mathcal{N}}, \ldots, h_n^{\mathcal{N}}\right\}$.Next, we need to provide context embeddings for the input of the decoder.

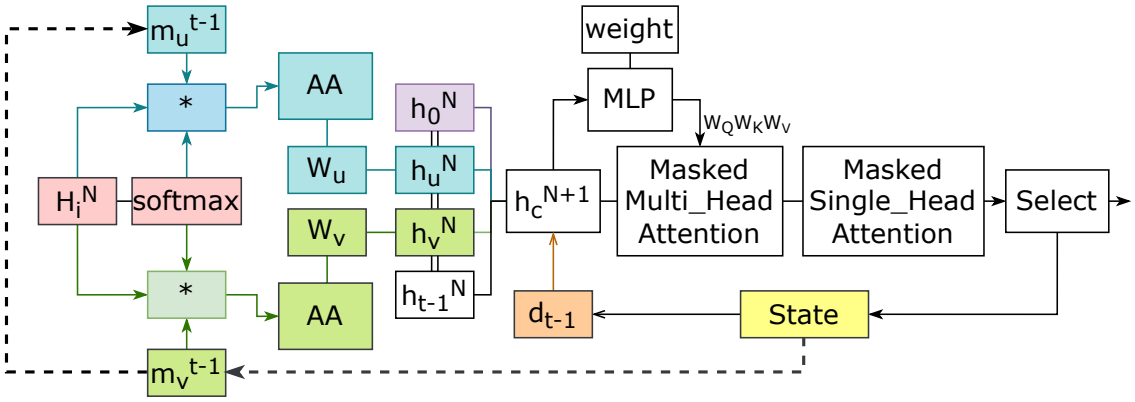

Figure 2: The colored sections represent the additional information we added to the context embedding.

Decoder: As shown in Fig. 2, the decoder is a recursive decoder that updates its current state with information related to the selected node. To implicitly capture dynamic information, we aim to pass updated information to the decoder through context embeddings. Previous context embeddings only utilized static information. For example, Kool et al. (2019) averaged all node embeddings to obtain a graph embedding $h_g^{\mathcal{N}} = \mathrm{mean}\left(h_0^{\mathcal{N}}, \ldots, h_n^{\mathcal{N}}\right)$, which was then aggregated with the current node embedding to form the context embedding. Our aim is to extract more effective information from the current state.

We observe that the mask in the state can serve as a dynamic source of information that can be utilized. Although the mask can be utilized to mask visited nodes in later attention layers, we aim to capture the information provided by the mask in advance to enhance the decoding ability of the decoder. To achieve this, we partition the graph embedding based

on the mask into two parts: the embedding of unvisited nodes and the embedding of visited nodes. Furthermore, previous studies have not considered the depot node as an input. We believe that the model should consider whether to return to the depot when selecting a path, rather than solely relying on exceeding the capacity limit to force vehicles to return to the depot. Therefore, we include the depot node embedding in the context embedding. We aggregate this information using an adaptive aggregation method and also add the dynamic information of the remaining capacity of the vehicles to the context embedding. Ultimately, we obtain a more comprehensive and rich dynamic context embedding.

The visited graph embedding is shown as follows:

$$H_v^{\mathcal{N}} = W_v \left(m_v^{t-1} * H_i^{\mathcal{N}} * \text{Softmax}\left(H_i^{\mathcal{N}}\right)\right) \tag{20}$$

Here, W_v is a trainable parameter matrix. m_v^{t-1} is a binary code that represents the masked state information of the graph. The visited nodes are marked as 1, and the unvisited nodes are marked as 0. AA represents Adaptive Aggregation, which is an aggregation method proposed by us for this dynamic graph embedding. The process is as follows:

$$h_v^{\mathcal{N}} = AA(H_v^{\mathcal{N}}) = \text{sum}(\left(\text{reshape}\left(\text{softmax}\left(W_A\left(\text{flat}\left(H_v^{\mathcal{N}}\right)\right)\right)\right)\right) * H_v^{\mathcal{N}}) \tag{21}$$

First, we flatten the $H_v^{\mathcal{N}}$, pass it through a fully connected layer to obtain a weight vector, and then reshape the weight vector to its original dimension. Finally, we perform aggregation through weighted summation. Similarly, the unvisited graph embedding $h_u^{\mathcal{N}}$ can obtain in this way. Finally, our context embedding can be defined as follows:

$$h_c^{\mathcal{N}} = W_c \,\text{concat}\left(\left(h_{\pi_{t-1}}^{\mathcal{N}} + h_0^{\mathcal{N}} + h_v^{\mathcal{N}} + h_u^{\mathcal{N}}\right), d_{t-1}\right) \tag{22}$$

Here, d_{t-1} represents the remaining capacity of the current vehicle and W_c is a trainable parameter matrix. The purpose of this matrix is to remap the merged context embedding back to the d_h dimension. Finally, we input the context embedding $h_c^{\mathcal{N}}$ into the decoder.

For the second part of the decoder, we utilize the model proposed by Lin et al. (2021). Specifically, they pass the weight vector through an hypernetwork to generate the parameters and for the multi-head attention layer. By connecting the weights and the decoder, we can generate different decoders by inputting different weights. Through this approach, a single model can approximate the Pareto set.

$$h_c^{\mathcal{N}+1} = MHA\left(Q = W_Q(\lambda)h_c^{\mathcal{N}}, K = W_K(\lambda)H_i^{\mathcal{N}}, V = W_V(\lambda)H_i^{\mathcal{N}}\right) \tag{23}$$

Here, Q, K, and V refer to the query, key, and value of the multi-head attention layer, respectively. The calculation method is similar to that described above, except that we need to mask the previously visited nodes by using a mask that satisfies two constraints: (1) the node has been visited before and (2) the demand of the next customer exceeds the remaining capacity of the current vehicle. Finally, a single-head attention layer is used to calculate the probability of selecting each city.

$$u_i = \begin{cases} C * \tanh\left(\dfrac{\left(h_c^{\mathcal{N}+1}\right)^T h_i^{\mathcal{N}}}{\sqrt{d}}\right) & , i \text{ is not masked} \\ -\infty & , \text{ otherwise} \end{cases} \tag{24}$$

$$p_{\theta(\lambda)}\left(\pi_t = i \mid \pi_{0:t-1}\right) = \frac{e^{u_i}}{\sum_{j=0}^{n} e^{u_j}} \tag{25}$$

We use C=10 Bello et al. (2017) to prune the final results. we choose the next customer node based on the calculated probabilities and update the state $s_{(t-1)}$ for the next decoding step. If the next node to visit is the depot, we reset the vehicle information and proceed to the next decoding step until all customer nodes have been visited. For a given instance s, the sequence solution can be obtained from the policy $p_{\boldsymbol{\theta}(\lambda)}(\boldsymbol{\pi} \mid s)$ based on the probability chain rule.

4.2. Optimization of the MOCOP

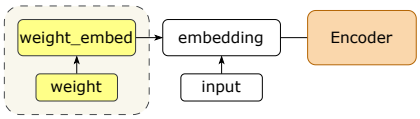

Figure 3: The yellow part is the module we added.

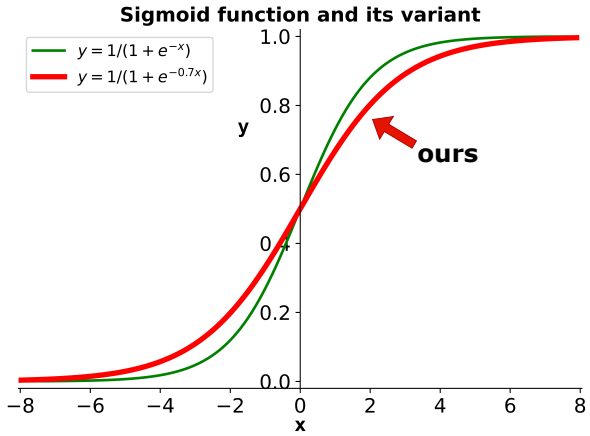

Figure 4: Sigmoid function and its variant.

Weight-encoder: To improve MOCOP, we propose a new approach for model enhancement. In the past, models acquired more models with different weights through transfer technology to generate a Pareto set. Hence, weight size was not significant for each model as they were only used to provide a weight for each objective in a decomposition-based approach to obtain a reward value. However, for approximating a Pareto set with a single model, we need to train the model with different weights simultaneously. Therefore, weight should also be integrated into the model as effective information. As depicted in Fig. 3, we add the weight as an input to the encoder to directly affect its output. This implies that different embedding results will be obtained for the same instance with different input weights. In contrast to other existing DRL solvers, where the encoder outputs are fixed, and the encoding results are the same for any weight, our encoder is more flexible, and dynamic, and can express more information.

To integrate weight information into the encoder, we do not use a complex network. Instead, we simply map the weight to an embedding through a simple MLP layer and then aggregate it with the node embedding. The specific implementation is as follows:

$$h_\lambda = w_\lambda^2\left(w_\lambda^1 \lambda^i + b_1\right) + b_2 \tag{26}$$

Where W_λ^1 and W_λ^2 are trainable parameter matrices and b_1 and b_2 are bias vectors. Therefore, the true input to the formula. 10 should be:

$$h_i^0 = h_i^0 + h_\lambda \tag{27}$$

Weight distribution: $weight = (\lambda_1, \lambda_2)$. Previously, weight distribution was generated based on uniform sampling. λ_1 was uniformly sampled between 0 and 1, and λ_2 was set to $1 - \lambda_1$. Under normal circumstances, extreme solutions like $(\lambda_1, \lambda_2) \in \{(0.1, 0.9), (0.9, 0.1)\}$ are difficult to train. However, based on the uniform sampling method, the weights have the same probability of occurrence, resulting in the same training degree of the model for all weights. Such a sampling method is not efficient enough. So we decide to design a more efficient weight sampling method. We find inspiration from the Sigmoid function. As shown in Fig. 4, we first uniformly sample x from -8 to 8, then convert it to y using the Sigmoid function. This method increased the probability of weights falling into the $(0, 0.2]$ or $[0.8, 1)$ intervals. However, the weight distribution obtained by sampling through the ordinary sigmoid function does not meet our expectations. To obtain a more satisfactory weight distribution, we fine-tune the Sigmoid function parameters to **0.7**, resulting in a smoother probability distribution for each weight. By changing the weight distribution through the Sigmoid sampling method, we have improved the efficiency and effectiveness of the model, making it better equipped to handle extreme solutions. This leads to higher quality solutions overall. We will temporarily name the weight generated through this method as **Sigmoid sampling**. The implementation is as follows:

$$\lambda_1 = \frac{1}{1 + e^{-0.7x}} \quad , \lambda_2 = 1 - \lambda_1 \tag{28}$$

The two improvements we proposed above may not be helpful for CVRP, but they perform well on MOCOP. They can effectively improve convergence and can be generalized to other MOCOP.

4.3. Multi-objective reinforcement learning

Given an instance s, the objective is to minimize the values of all weights:

$$\mathcal{J}(\boldsymbol{\theta} \mid s) = \mathbb{E}_{\lambda \sim \Lambda, \boldsymbol{\pi} \sim p_{\boldsymbol{\theta}(\lambda)}}(\boldsymbol{\pi} \mid s) f^{ws}(\boldsymbol{\pi} \mid \lambda) \tag{29}$$

Where Λ is the Sigmoid distribution that we proposed.

$$\nabla \mathcal{J}\left(\boldsymbol{\theta} \mid \lambda^j, s\right) = \mathbb{E}_{\boldsymbol{\pi} \sim p_{0(\lambda)}}(\pi \mid s) \left[\left(f^{ws}\left(\boldsymbol{\pi} \mid \lambda^j, s\right) - b\left(s \mid \lambda^j\right)\right) \nabla_{\boldsymbol{\theta}(\lambda^j)} \log p_{\boldsymbol{\theta}(\lambda^j)}(\boldsymbol{\pi} \mid s)\right] \tag{30}$$

The term $b\left(s \mid \lambda^j\right)$ represents the baseline expected reward used to reduce the variance of the expectation. This gradient is estimated through Monte Carlo sampling Beasley and Martin (1984). In each training cycle, a weight λ^j is randomly agsigned to each batch instance $\{s_1, \ldots, s_B\} \sim S$, with $\lambda^j \epsilon \left\{\lambda^1, \ldots, \lambda'\right\} \sim \Lambda$. The customer nodes of each instance are sequentially visited as the first node to create a circuit, enforcing different circuits [43]. Thus, n distinct circuits $\left\{\pi_i^1, \ldots, \pi_i^n\right\} \sim p_{\boldsymbol{\theta}(\lambda)}\left(\boldsymbol{\pi} \mid s_i\right)$ are formed, and the

expected reward is approximated by the mean reward of the n circuits, given by $\left(s_i \mid \lambda^j\right) = \frac{1}{N}\sum_{n=1}^{N} f^{Ws}\left(\boldsymbol{\pi}_{ji}^n \mid \lambda^i, s_i\right)$. The gradient approximation is as follows:

$$\nabla \mathcal{J}(\boldsymbol{\theta}) \approx \frac{1}{JBN}\sum_{j=1}^{J}\sum_{i=1}^{B}\sum_{n=1}^{N}\left[\left(f^{ws}\left(\boldsymbol{\pi}_{ji}^n \mid \lambda^j, s_i\right) - b\left(s_i \mid \lambda^j\right)\right)\nabla_{\boldsymbol{\theta}(\lambda)} \log p_{\boldsymbol{\theta}(\lambda^j)}\left(\boldsymbol{\pi}_i^n \mid s_i\right)\right] \quad (31)$$

5. Experiments: MOCVRP

In this section, we will provide a brief overview of the model and some parameter settings used during training. We trained our model on a Geforce RTX 3090.

5.1. Experimental settings

MOADAM Hyperparameters: We compared our trained MOADAM model with other frameworks that use DRL attention models. Therefore, the hyperparameters of our model were designed based on AM Kool et al. (2019). The dimension d_h of node embeddings was set to 128. The number of attention layers N in the encoder was 6. For each multi-head attention module in each layer, the number of heads M was set to 8.

Training Dataset: We used MOCVRP instances with $3d$ inputs $[x, y, d]$ as our training data. For problems with 20, 50, and 100 customer nodes, we set the capacity of the vehicles to 30, 40, and 50, respectively, to be consistent with some prior works. The coordinates $[x, y]$ of any node were generated from a uniform distribution over $[0, 1]$, while the demand d of customer nodes was randomly sampled from integers in the range $[1, 10]$.

Training Parameter Settings: The training period of the model was 200 epochs. For each batch size during training, we randomly generated a weight. We used the Adam optimizer with a learning rate of $\eta = 10^{-4}$and weight decay is 10^{-4}.

PF Subproblems: We set the number of weight vectors N in the Pareto front construction to 101, uniformly distributed between (0,1). Specifically, $\lambda_1 = (1, 0)$, $\lambda_2 = (0, 1)$.

Evaluation Metrics: We used Hypervolume (HV), Gap, and C-metric to compare the performance of different algorithms. HV is an important metric that evaluates both the convergence and diversity of the PF. For 20, 50, and 100 nodes, we set the reference point to [15,3], [40,3], [60,4]. The gap is the ratio of the difference in HV relative to the best result. C-metric means solution set coverage.

We compared our trained **MOADAM** with classic solvers **NSGA-II** Deb et al. (2002), **MOEA/D** Zhang and Li (2007) and other advanced frameworks that use DRL attention models. We use **AM-MOCO** proposed by Li et al. (2021) to train multiple AM models for different weights. We first train a single AM model with weights [0,1] for 100 epochs, then transfer its parameters to models of adjacent sub-problems over five epochs, and so on to obtain models with different weights. **ML-AM** proposed by Zhang et al. (2022) generated a meta-model by transferring parameters for different weights, then fine-tuned the meta-model for 10 steps to obtain sub-models for different weights. **P-MOCO** proposed by Lin et al. (2021) used a hypernetwork to directly generate a Pareto set through a single model.

5.2. Experimental result

We further evaluated the performance of our model on MOCVRP. MOCVRP20, MOCVRP50, and MOCVRP100 models were trained on instances with 20, 50, and 100 nodes. The experimental results are shown in Table.1.

Table 1: Experimental results on two-objective MOCVRP instances.

Methods	MOCVRP20		MOCVRP50		MOCVRP100	
	HV	Gap	HV	Gap	HV	Gap
NSGAII	9.420	14.13 %	26.846	19.48 %	36.323	22.10 %
MOEA/D	8.971	19.44 %	25.388	26.34 %	35.865	23.66 %
AM-MOCO(101 models)	10.449	2.54 %	30.924	3.84 %	42.552	4.36 %
ML-AM(101 weights)	10.256	4.48 %	30.336	5.85 %	41.526	6.93 %
P-MOCO(101 weights)	10.561	1.45 %	31.692	1.33 %	43.831	1.27 %
MOADAM(101 weights)	**10.715**	**0.0%**	**32.076**	**0.0%**	**44.352**	**0.0%**

Our proposed model, **MOADAM**, outperforms other models in solving problems with different numbers of nodes. For **AM-MOCO**, only 5 epochs to transfer to the new weight sub-problem may not be perfectly matched, but transferring 100 different weights already requires 600 epochs, if you continue to increase the transfer epoch will cost too much time. For **ML-AM**, the results were slightly worse as it only fine-tuned for 5 iterations, but it required much less transfer time than **AM-MOCO**. However, training the meta-model required a lot of time as it needed to fine-tune for various weights. Currently, **P-MOCO** is considered the most advanced and effective method, as demonstrated in Table 2 and Fig.5. We compare our results with **P-MOCO** to highlight the effectiveness of our improvement.

Table 2: Evolutionary history between MOADAM and P-MOCO.

Methods	MOCVRP20			MOCVRP50			MOCVRP100		
	HV	f_1	f_2	HV	f_1	f_2	HV	f_1	f_2
P-MOCO	10.561	6.1866	1.7783	31.692	10.8511	1.9057	43.831	16.4445	1.9893
ADAM(Ours)	10.674			31.983			44.244		
MOADAM(Ours)	**10.715**	**6.0628**	**1.7781**	**32.076**	**10.5533**	**1.905**	**44.352**	**15.9905**	**1.9890**

We conduct a comparison between our proposed model **MOADAM** and **P-MOCO**, and validate the effectiveness of our model improvements by gradually incorporating different modules. Initially, we focus on the improvement of CVRP and named the enhanced model **ADAM**. As shown in Table.2, our proposed model significantly outperform **P-MOCO** in HV for varying numbers of nodes. Subsequently, we introduce the optimization of MOCOP, resulting in the formulation of our final model, **MOADAM**. The results show that **MOADAM** generates better HV values than **ADAM**, indicating that improvements in both CVRP and MOCOP are feasible solutions for enhanced results.

To evaluate the effectiveness of our model, we perform a single-objective CVRP with weights set to $(1, 0)$ and $(0, 1)$. The results demonstrate that our proposed model outperforms **P-MOCO** in terms of both f_1 and f_2, particularly f_1. Furthermore, we generate a Pareto set using our proposed model and **P-MOCO**, as depicted in Fig. 5. Our generated Pareto set is superior to **P-MOCO** in terms of diversity and convergence. To further vali-

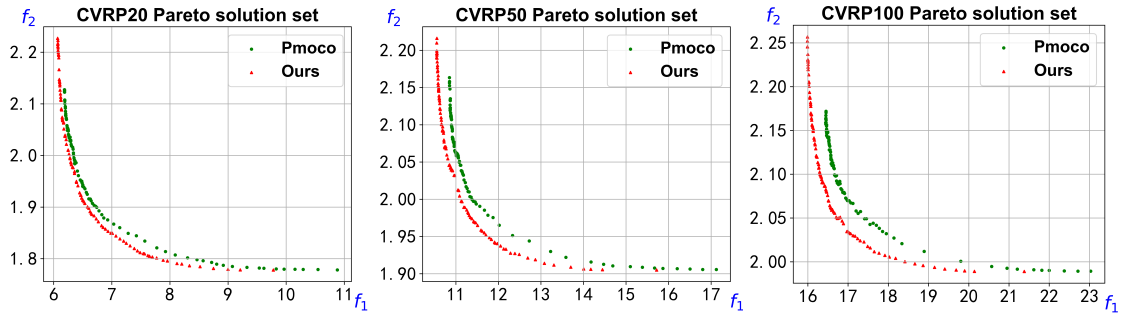

Figure 5: Comparison between the Pareto set of our model and P-MOCO.

date this, we utilize the evaluation metric C-metric, obtaining $C\left(MOADAM, PMOCO\right) = 1$ and $C\left(PMOCO, MOADAM\right) = 0$ for instances with varying numbers of nodes. These results demonstrate that all solutions in **P-MOCO** can be dominated by some solutions in our solution set, whereas none of the solutions in our set are dominated by them. This further supports our claim that our proposed model, **MOADAM**, outperforms the current state-of-the-art **P-MOCO** in terms of solution quality.

6. Conclusion

This paper proposes a novel model, MOADAM, based on reinforcement learning for MOCVRP. In CVRP, we improve the model by reordering the encoder and enhancing the application of edge information in the encoder part, adding more dynamic information to refine context embeddings. This improvement is also applicable to single-objective CVRP. In MOCOP, we design a new weight sampling method to enable the model to better solve the extreme solution and integrate weight information into the input embedding of the encoder through a simple MLP network. The encoder will be fine-tuned to accommodate different weights. This improvement is a new idea put forward in the paper and can be applied to other types of MOCOP. Overall, our model is more flexible and outperforms existing methods in terms of performance. In the future, we plan to explore more multi-objective combinatorial optimization problems.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grant 62176161, and the Scientific Research and DevelopmentFoundations of Shenzhen under Grant JCYJ20220818100005011.

References

Enrique Angel, Evripidis Bampis, and Laurent Gourvès. *A dynasearch neighborhood for the bicriteria traveling salesman problem*, pages 153–176. Springer, 2004.

John E Beasley and Richard R Martin. Monte carlo approach to vehicle routeing. *European Journal of Operational Research*, 16(2):260–266, 1984.

Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. In *Proc. ICLR (Workshop)*, pages 1–16, 2017.

Juan Castro-Gutierrez, Dario Landa-Silva, and Jose M Perez. Nature of real-world multi-objective vehicle routing with evolutionary algorithms. In *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, pages 257–264, Oct. 2011.

Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T Meyarivan. A fast and elitist multi-objective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

M. Desrochers, Y. Dumas, and M.M. Solomon. A branch-and-price algorithm for the vehicle routing problem with time windows. *Operations Research*, 40(2):342–354, 1992.

Matthias Ehrgott. *Multicriteria optimization*, volume 491. Springer Science & Business Media, 2005.

F. Focacci, A. Lodi, and M. Milano. An exact algorithm for the vehicle routing problem with time windows and simultaneous delivery and pickup. *Transportation Science*, 41(2): 253–264, 2007.

John J Hopfield and David W Tank. 'neural' computation of decisions in optimization problems. *Biol. Cybern.*, 52(3):141–152, 1985.

Andrzej Jaszkiewicz. Genetic local search for multi-objective combinatorial optimization. *European Journal of Operational Research*, 137(1):50–71, 2002.

Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *Proc. ICLR*, 2019.

Kai Li, Tao Zhang, and Rui Wang. Deep reinforcement learning for multiobjective optimization. *IEEE Trans. Cybern.*, 51(6):3103–3114, Jun. 2021.

X. Lin, Z. Yang, and Q. Zhang. Pareto set learning for neural multi-objective combinatorial optimization. In *International Conference on Learning Representations*, 2021.

Kaisa Miettinen. *Nonlinear multiobjective optimization*. Springer Science & Business Media, 2012.

Meysam Nazari, Ahmad Oroojlooy, Lawrence Snyder, and Martin Takác. Reinforcement learning for solving the vehicle routing problem. In *Proc. Adv. Neural Inf. Process. Syst.*, pages 9839–9849, 2018.

S. Ropke and D. Pisinger. An exact algorithm for the vehicle routing problem with time windows and multiple routes. *Transportation Science*, 40(3):429–443, 2006.

A. Subramanian and S.V. Ukkusuri. An exact algorithm for the vehicle routing problem with time windows and limited vehicle capacity. *Transportation Research Part B: Methodological*, 67:166–182, 2014.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, volume 27, pages 3104–3112, 2014.

Anupam Trivedi, Dipti Srinivasan, Krishnendu Sanyal, and Abhiroop Ghosh. A survey of multi-objective evolutionary algorithms based on decomposition. *IEEE Transactions on Evolutionary Computation*, 21(3):440–462, 2016.

Ashish Vaswani et al. Attention is all you need. In *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, pages 5998–6008, 2017.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, volume 28, pages 2692–2700, 2015.

Jiahai Wang, Ying Zhou, Yong Wang, Jun Zhang, CL Philip Chen, and Zibin Zheng. Multiobjective vehicle routing problems with simultaneous delivery and pickup and time windows: formulation, instances, and algorithms. *IEEE transactions on cybernetics*, 46 (3):582–594, 2015.

Adams Wei Yu, David Dohan, Quoc Le, Thang Luong, Rui Zhao, and Kai Chen. Fast and accurate reading comprehension by combining self-attention and convolution. In *Proc. Int. Conf. Learn. Represent. (ICLR)*, pages 1–15, 2018.

Qingfu Zhang and Hui Li. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731, 2007.

Zishuo Zhang, Zhigang Wu, Hua Zhang, and Jian Wang. Meta-learning-based deep reinforcement learning for multiobjective optimization problems. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.