廣東工業大學

本科毕业设计（论文）

基于深度强化学习的车辆路径规划方法

学　　院　计算机学院

专　　业　人工智能

年级班别　2021 级（1）班

学　　号　3121005341

学生姓名　黄志鹏

指导教师　李亦卿

2025 年 6 月

摘要

随着电子商务和智能物流以及全球化供应链的蓬勃发展，车辆路径规划（Vehicle Routing Problem, VRP）成为了提升运输效率以及降低运输成本的核心问题之一。作为 VRP 的经典变种问题之一的带容量约束的车辆路径问题（Capacitated Vehicle Routing Problem, CVRP）因为其 NP-Hard 的特性，使用传统方法如精确算法，会导致计算复杂度过高。启发式算法又容易陷入局部最优解。因此，探索基于深度强化学习的 CVRP 求解方法具有重大的研究意义。

本文提出了一种结合了分治思想和近端策略优化（Proximal Policy Optimization, PPO）算法的方法，并且使用了图神经网络（Graph Neural Network, GNN）作为主题框架，进行客户状态建模，其目的在于高效的求解中大规模的 CVRP 问题。首先构建了分治框架，通过 GNN 编码客户节点的空间与需求特征，然后生成聚类划分策略，将全局的 CVRP 问题分解为多个满足容量约束的旅行商问题。随后利用 OR-Tools 工具并行求解子问题。PPO 算法通过奖励函数驱动策略网络去优化聚类方案，从而形成了分解-求解-反馈的闭环训练机制。

仿真测试阶段，使用基于 PyTorch 的框架搭建了包含图嵌入网络、参数化预测网络和 PPO 策略网络的强化学习模型，结合了多级并行架构加速求解。实验表面，所提方法能够有效的降低路径总成本。

关键词：车辆路径规划，强化学习，图神经网络，智能物流

注：本设计（论文）选题来源于国家自然科学青年基金项目（62102096）。

Abstract

With the rapid development of e-commerce, intelligent logistics, and the globalized supply chain, the Vehicle Routing Problem (VRP) has become a key challenge in improving transportation efficiency and reducing operational costs. Among its many variants, the Capacitated Vehicle Routing Problem (CVRP), which involves capacity constraints, is particularly complex due to its NP-Hard nature. Traditional exact algorithms often suffer from high computational complexity, while heuristic methods tend to get trapped in local optima. Therefore, it is of great research significance to explore deep reinforcement learning (DRL)-based approaches for solving the CVRP.

In this thesis, A novel method is proposed that combines the idea of problem partitioning with the Proximal Policy Optimization (PPO) algorithm, and employs a Graph Neural Network (GNN) for effective state representation. The proposed method is designed to efficiently solve medium and large-scale CVRP instances. Firstly, a divide-and-conquer framework was constructed. By using GNN to encode the spatial and demand features of customer nodes, a clustering partition strategy was generated to decompose the global CVRP problem into multiple TSP problems that satisfy capacity constraints. Subsequently, the OR-Tools tool was utilized to solve the sub-problems in parallel. The PPO algorithm, driven by a reward function, optimized the clustering scheme through the policy network, thus forming a closed-loop training mechanism of decomposition-solution-feedback.

In the simulation test stage, a reinforcement learning model was built using a PyTorch-based framework, which included a graph embedding network, a parameterized prediction network, and a PPO policy network. A multi-level parallel architecture was combined to accelerate the solution. Experiments demonstrated that the proposed method could effectively reduce the total path cost.

Key words: Vehicle routing planning, reinforcement learning, graph neural networks, intelligent logistics

目录

1 绪论

1.1 课题研究背景

随着电子商务迅速普及，全球供应链变得越来越复杂。我们可以看到，物流公司的配送效率正逐渐成为企业竞争力的关键因素之一。国际物流协会在 2024 年的报告中指出，全球物流市场的规模已经突破 12 万亿美元。就成本结构来说，最后一公里的配送部分占了整个物流成本的 28%以上，这一比例非常惊人。

拿配送效率来说，车辆路径规划问题（Vehicle Routing Problem, VRP）一直是优化物流的核心问题。VRP 的目标，是为每辆车安排一条高效又省钱的配送路线。本文希望能满足客户的需求，并且尽可能降低运输成本。

但是，VRP 不是一个容易解决的问题。虽然已经有很多经典算法被提出，像一些启发式方法和精确算法，但真正做到又快又准，还是很有挑战性的任务。就 CVRP（带容量约束的车辆路径问题）来说，它是 VRP 中最常见的一个变种。它的核心难点，在于如何在车辆载重有限的前提下，把客户订单分配得更合理，并且规划出一条最省距离的路线。

本文面对的最大问题是，这类问题本身属于 NP-Hard 问题。也就是说，问题规模一旦扩大，计算难度就会迅速上升。比如，使用分支定界法来求解 100 个客户点的 CVRP 问题，可能需要好几个小时，甚至几天。这对于实际物流调度来说，基本无法接受。

为了应对这种复杂性，很多人开始转向启发式算法。比如模拟退火和遗传算法，它们能在短时间内给出还不错的解决方案。但是它们也有缺点，表现常常依赖于专家经验，而且很容易陷入局部最优。拿遗传算法来说，如果初始种群质量不好，最终得到的结果可能只是一个次优解，而不是真正想要的最优路径。

总的来说，CVRP 虽然是一个被广泛研究的组合优化问题，但它在实际应用中仍然非常有挑战性。但是近些年来，深度强化学习在解决复杂的决策问题时展现出了自己的独特优势。深度强化学习通过智能体与环境的交互学习策略，能够从高纬状态空间中提取出有效的特征，并且能够在动态环境中自适应优化决策。这一特性使得其在组合优化领域受到广泛的应用。但是在 CVRP 这一类复杂约束的问题中仍然面临着诸多挑战。例如，车辆容量约束需要更加精确和细致的状态表示以及奖励函数如何进行设定等挑战。

因此，对基于深度强化学习的 CVRP 求解办法的探索，不仅仅是对传统技术的补充，同样也是推动深度强化学习发展的重要研究方向。

1.2 课题研究意义

该课题的研究具有理论意义和实践意义这两方面，首先是理论意义包括但不限于以下两点：

首先是扩展了深度强化学习在组合优化中的应用领域。CVRP 作为 NP-hard 问题，其求解需要同时处理离散决策和连续状态空间，同时需要求解两种问题，这对深度强化学习算法的设计提出了新的挑战。通过研究基于深度强化学习的 CVRP 求解框架，可以深化对神经网络建模复杂约束的理解。通过图神经网络(GNN)编码客户-车辆关系，分层处理大规模的 CVRP 问题等等。这些探索都将丰富深度强化学习在组合优化领域的理论体系。

其次是促进了跨学科方法的融合。传统运筹学与人工智能的结合是当下研究的热点。本课题通过将深度强化学习与经典启发式规则结合，可探索“学习+搜索”的混合优化范式，为复杂约束下的路径规划问题提供新的方法。

本课题同样也具有实践意义，包括但不限于以下两点：

首先是提升物流行业效率和可持续性发展。高效的 CVRP 求解方案可显著降低车辆的空车驾驶率，减少燃油消耗从而减少碳排放。例如据国际能源署统计[1]，全球物流运输碳排放占比超过 20%，通过优化路径规划可降低 5%-15%的运输成本。对于电商、零售等依赖高频配送的行业，本研究成果可直接提升其供应链响应速度与客户满意度。

其次是支持智慧城市与智能交通系统的建设。在车联网与自动驾驶技术高速发展的大背景下，运载工具能够进行实时的动态路径规划成为了该大背景下的核心需求。基于深度强化学习的 CVRP 能够适应实时交通流量变化和突发订单需求，为无人配送车、无人机等新型运载工具提供底层决策支持。

1.3 课题研究现状

在过去的几十年中，人们已经开发了多种方法来解决组合优化问题，许多人也尝试通过使用人工智能来求解组合优化问题。第一次尝试是由 Vinyals 等人提出的[2]。他们引入了指针网络的概念，这是一种受到序列到序列模型启发的模型而创造的模型。由于

指针网络对编码器序列的长度是固定的，这一特性使该模型能够应用于组合优化问题。其中输出序列长度是由源序列确定的。他们以监督学习的方式训练并使用指针网络架构，从地面实况最优（或启发式）解决方案中找到接近最优的旅行商问题（Travelling Salesman Problem, TSP）[13]。这种对监督学习的依赖性阻碍了指针网络找到比训练期间提供解决方案更好的解决方案。为了解决这个问题，Bello 等人通过开发一个神经组合优化框架来解决这个问题[3]，该框架使用强化学习来优化由指针网络建模的策略。他们使用了几个经典的组合优化问题（如 TSP 和背包问题）来测试他们的模型架构。测试结果显示了其体系架构的有效性和通用性。在相关问题上，Dai 等人[5]使用图嵌入结构[5]和深度 Q 学习（Deep Q Network, DQN）算法[6]解决图上的优化问题。尽管 VRP 可以用一个带有加权节点和边的图来表示，但是他们提出的模型并不直接适用，原因是在 VRP 中，一个特定的节点（例如仓库）可能会被访问多次。

1.4 论文主要内容

本篇论文将完成以下内容：

第一章 绪论：介绍课题研究背景，阐明课题研究意义，对国内外研究现状进行简单叙述。

第二章 相关技术和原理：对车辆路径规划的系统模型进行介绍。对图神经网络进行简要介绍。对强化学习算法的原理进行介绍，以及对近端策略优化算法的有关概念进行说明解释和公式推导。

第三章 问题建模与求解：将定义问题求解的框架，已经各个不同模块的具体构建与求解。

第四章 仿真测试：使用 python 平台对基于深度强化学习的车辆路径规划问题进行仿真和测试。

2 相关技术和原理

2.1 旅行商问题定义

旅行商问题（Traveling Salesman Problem, TSP）是组合优化和计算复杂性理论中最著名的问题之一，属于非确定性多项式困难问题（NP-Hard）。它不仅在理论计算机科学中具有重要地位，同时在物流、制造、生物信息学等领域也具有广泛应用。

TSP 可以建模为一个完全加权图：

1. 顶点集（V）：表示城市的集合，即$V = \{v_1, v_2, \dots, v_n\}$。
2. 边集（E）：表示所有城市之间的连接，即$E = \{(v_i, v_j) | v_i, v_j \in V, i \neq j\}$。
3. 权重（d）：每条边(v_i, v_j)有一个非负权重d_{ij}，表示城市v_i和v_j之间的距离。

TSP 的优化目标是找到一个哈密尔顿回路（Hamiltonian Cycle），即：

1. 经过每个城市恰好一次（即一个排列$\pi = (\pi_1, \pi_2, \dots, \pi_n)$）。
2. 最后返回起点，形成一个闭合路径。
3. 使得总成本最小：

$$\text{minimize} \sum_{i=1}^{n-1} d_{\pi_i, \pi_{i+1}} + d_{\pi_n, \pi_1}。 \tag{2.1}$$

其主要变体有对称旅行商问题（Symmetric TSP, STSP），非对称旅行商问题（Asymmetric TSP, ATSP），度量旅行商问题（Metric TSP）等。

2.2 车辆路径规划问题定义

车辆路径规划问题是一类经典的组合优化问题，其核心目标是再考虑一些约束的情况下，以用户定义的目标从仓库服务于所有客户的需求，设计最优的车辆配送路线，以最小化运输成本（如总行驶距离、时间或车辆数）。在本文中，我主要研究带容量约束的车辆路径问题（Capacitated Vehicle Routing Problem, CVRP），其具体定义如下[7]：

给定一个客户点集$\Omega = \{I_0, I_1, \dots, I_n\}$，其中$I_0$代表仓库，是所有车辆的起点和终点。$I_i$代表第$i$个客户。每个客户$I_i = (x_i, y_i, d_i)$包含三个特征：$x$坐标$x_i$，$y$坐标$y_i$，以及客户需求$d_i$。

CVRP 的优化目标是在以下的约束条件下，找到一组最优的车辆路径：

1. 车辆容量约束：每辆车的总载货量不得超过其容量C，即对于任意一条路径π_k，其服务的客户需求总和满足$\sum_{l_i \in \pi_k} d_i \leq C$。

2. 车辆数量约束：可用车辆的最大数量为l_m，即路径总数不超过l_m。

3. 访问约束：每个客户必须被恰好访问一次，且所有客户需求必须被满足。

问题的数学优化目标表达式可以表示为：

$$\min \sum_{k=1}^{l} L(\pi_k), \tag{2.2}$$

其中l是实际使用的车辆数量（$l \leq l_m$），π_k表示第k条路径的客户序列，$L(\pi_k)$表示路径π_k的欧式距离长度。

CVRP 的 NP-Hard 性质可以通过以下两点进行证明：

1. 规约至 TS：当车辆数$m = 1$且容量$\sum_{l_i \in \pi_k} d_i \leq C$时，CVRP 退化成 TSP。已知 TSP 是 NP-Hard 问题[16]，因此 CVRP 至少与 TSP 同等复杂。

2. 归约至装箱问题：若固定成本路径$c_{ij} = 1$，CVRP 的目标转化为最小化车辆数m，等价与将客户需求装入容量为C的箱子中。装箱问题也已经被证明为 NP-Hard 问题[17]，因此 CVRP 的决策，即判定是否存在m条可行路径数属于 NP-Hard 类。

2.3 图神经网络

图神经网络是一种专门处理图结构数据的深度学习模型。与传统的神经网络（如卷积神经网络，循环神经网络）不同。图神经网络的优势在于能够有效的捕捉到非欧几里得数据（如图数据，社交网络，分子结构）中的复杂的拓扑结构关系。图数据有两个核心组件：节点和边。节点表示实体（如用户、分子中的原子等）。边表示实体之间的关系（如社交关系、化学键等）。图神经网络的核心目标是通过聚合邻接节点之间的信息，生成每一个节点的低维向量表示，从而能够支持节点分类、图分类等任务。图神经网络有几个关键的特点：

1. 非欧几里得结构：图数据没有固定的网络结构，节点间的连接关系动态且不规则。

2. 局部依赖性：节点的属性不仅仅取决于自身，还受其邻居节点的影响。

3. 可扩展性：图神经网络需要处理从数百到数亿个节点的大规模图数据。

2.2.1 核心算法

图神经网络的核心思想是通过迭代的消息传递更新节点表示，每一层的图神经网络的操作可以分为三步：首先是消息生成，每个节点生成需要发送给邻居的信息。然后是消息聚合，节点收集并且聚合邻居的信息。最后是状态更新，结合自身状态和聚合信息生成新的表示。其数学公式表示为：

$$h_v^{(l+1)} = \phi\left(h_v^{(l)}, \bigoplus \psi\left(h_v^{(l)}, h_u^{(l)}, e_{uv}\right)_{u\in\mathcal{N}(v)}\right), \tag{2.3}$$

其中，$h_v^{(l)}$表示节点v在第l层的表示，$\mathcal{N}(v)$是邻居节点，ϕ和ψ是可学习的函数，$\oplus$ 表示为聚合操作（求和、均值、最大值等）。

2.2.2 经典模型

首先在图神经网络中最经典的模型之一就是图卷积网络（Graph convolutional network, GCN）。GCN 通过谱域卷积定义图卷积操作。其传播的公式为：

$$H^{(l+1)} = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right), \tag{2.4}$$

其中，$\hat{A} = A + I$为添加自连接的邻接矩阵，$\hat{D}$为度矩阵，$W^{(l)}$为可训练的参数，σ为激活函数。GCN 的局限性在于无法动态进行调整邻居的权重。

在图神经网络中还有一个经典的模型是图注意力网络（Graph attention network, GAT)。GAT 通过注意力机制为不同邻居分配重要性权重。注意力系数计算为：

$$\alpha_{ij} = \frac{\exp\left(LeakyReLU(a^T[Wh_i||Wh_j])\right)}{\sum_{k\in\mathcal{N}(i)}\exp\left(LeakyReLU(a^T[Wh_i||Wh_k])\right)}。 \tag{2.5}$$

节点更新公式为：

$$h_i^{(l+1)} = \sigma\left(\sum_{j\in\mathcal{N}(i)}\alpha_{ij}Wh_j^{(l)}\right)。 \tag{2.6}$$

2.2.3 应用场景

图神经网络有诸多的应用场景。首先是社交网络的分析，如节点分类（预测用户的属性）、社区检测（发现社交网络中的潜在群体）、谣言检测（通过传播路径识别虚假信息）。其次可以进行交通预测，基于历史的数据预测未来交通的拥堵情况，还可以优

化实时导航路径。同时图神经网络在知识图谱领域也有成效，可以用于关系推理，如补全缺失的三元组，或进行实体链接，将文本中的实体与知识库中的条目进行对齐。

2.4 强化学习

2.2.1 强化学习概念

强化学习（Reinforcement Learning, RL）是机器学习的一个重要分支，其核心思想是通过智能体（Agent）与环境（Environment）的持续交互来学习最优决策策略。相比于监督学习需要大量标注数据，使得模型进行训练。强化学习则是采用类似于试错一样机制，通过奖励（Reward）来指导智能体的行为，并且优化智能体的行为。经典的强化学习可以总结为图 2.1 所示。

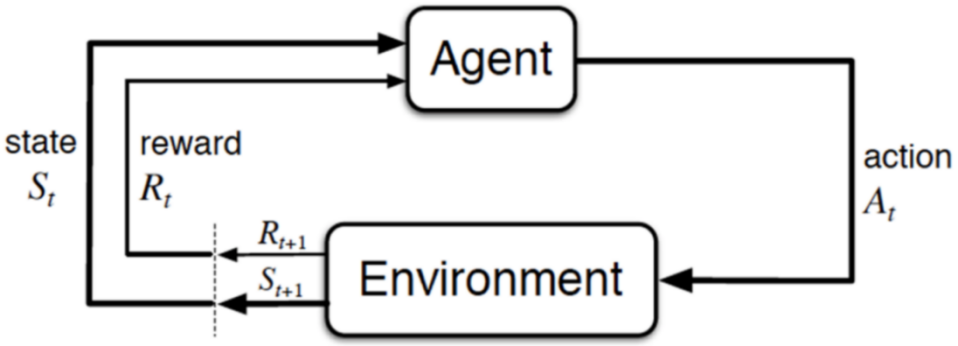

图 2.1　强化学习过程

图 2.1 中包括了强化学习中的核心四要素，即状态空间、动作空间、策略以及奖励函数。

状态空间$\mathcal{S}$是马尔可夫决策过程（Markovian Decision Process, MDP）中所有可能状态的集合，同时其满足马尔可夫性质。用数学公式的表达如下：

$$\mathbb{P}(s_{t+1}|s_t, a_t) = \mathbb{P}(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \dots), \tag{2.7}$$

动作空间$\mathcal{A}$是在给定状态$s \in \mathcal{S}$下可执行动作的集合，可分为：

$$\mathcal{A} = \begin{cases} a_1, \dots, a_k & \text{离散动作空间} \\ \mathbb{R}^{\mathrm{m}} & \text{连续动作空间} \end{cases}。 \tag{2.8}$$

策略π是从状态空间到动作空间的映射，可分为：

$$\pi(a|s) = \begin{cases} 1 & \text{确定性策略} \\ \mathbb{P}(a|s) & \text{随机性策略} \end{cases}。 \tag{2.9}$$

奖励函数$r: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$给出状态转移的即使评估，需要满足：

$$R_t = r(s_t, a_t, s_{t+1})。 \tag{2.10}$$

这四要素构成了强化学习中最基础的数学框架，任何强化学习的算法都需要明确定义这四要素及其的相互关系。

进一步，强化学习的目的是为了得到最优的策略从而获取最大化的奖励，因此衍生出了两种不同的方法：

1. 基于值函数的方法：通过求解一个状态或者状态下某一个动作的估值作为手段，从而寻找最佳的价值函数，找到了价值函数之后，再去选取最佳的策略。

2. 基于策略的方法：先进行策略评估，即对目前已经搜索到的策略函数进行估值，得到估值后，进行策略改进，然后不断重复这两步直到策略收敛。

2.2.2 Q-Learning

Q-Learning 是强化学习中基于值函数的算法，Q即为$Q(s,a)$，就是在某一个时刻的s（$s \in S$）状态下采取动作$a(a \in A)$能够获得收益的期望。环境回根据智能体的动作反馈相应的回报（reward）。该算法的主要思想就是将状态和动作构建成一张 Q-table 来储存Q值，然后根据Q值来选取能够获得最大回报的动作。

Q-Learning 的主要优势就是使用了时间差分法能够进行离线学习，使用了贝尔曼（Bellman）方程对马尔可夫过程求解最优策略。

通过 Bellman 方程求解马尔可夫决策过程的最佳决策序列，状态值函数$V_\pi(s)$可以评价当前状态的好坏，每个状态的值不仅仅由当前状态决定，同时也由后续的状态决定。因此状态的累计奖励求期望就可以得到当前状态s的状态值函数$V_\pi(s)$。Bellman 方程如下：

$$V_\pi(s) = E_\pi[R_{t+1} + \gamma V(s') | S_t = s], \tag{2.11}$$

最优累计期望可以用$V^*(s)$表示，可知最优值函数就是$V^*(s) = \max_\pi V_\pi(s)$，即：

$$V^*(s) = \max_\pi E[\sum_{t=0}^{H} \gamma^t R(S_t, A_t, S_{t+1}) | \pi, s_0 = s]。 \tag{2.12}$$

其次$Q(s,a)$状态动作值函数为：

$$q_\pi(s,a) = E_\pi[G_t | A_t = a, S_t = s], \tag{2.13}$$

其中$G_t = \{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots\}$是$t$时刻开始的总折扣奖励，从这里我能看出$\gamma$衰变

值对Q函数的影响，γ越接近 1 代表它越有远见会着重考虑后续状态的价值，当γ越接近 0 代表它越近视只会考虑当前利益的影响。最优价值动作函数$Q^*(s,a) = \max_\pi Q^*(s,a)$，打开期望如下：

$$Q^*(s,a) = \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q^*(s',a')。 \tag{2.14}$$

由此得到 Bellman 方程如下：

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a)\, V^*(s')。 \tag{2.15}$$

其中$Q^*(s,a)$是在最优策略下状态s采取动作a的最优Q值。$R(s,a)$是在状态s下采取动作a获得的即时奖励。γ是折扣因子。$P(s'|s,a)$是从状态s采取动作a转移到状态s'的概率。$V^*(s')$是在最优策略下状态s'的最优价值。

得到了 Bellman 方程后，然后就可以得到 Q-Learning 的核心公式，即更新Q值的公式，该公式是基于 Bellman 方程：

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)], \tag{2.16}$$

其中$Q(s,a)$是在状态s下采取动作a的Q值。α是学习率，控制新估计值和旧估计值之间的权衡。r是在执行了动作a后获得的及时奖励。γ是折扣因子。s'是执行动作a后观察到的新状态。a'是在新状态s'下选择的下一个动作。得到公式（2.15）后，即可以开始执行 Q-Learning 算法。

首先 Q-Learning 算法的第一步是初始化Q值。在算法开始时，Q值通常被初始化为零或者是一个很小的随机数，从而表示智能体对当前环境的无知。随后这个Q值将随着智能体与环境的交互而不断地更新和改进。

随后，在每一个时间步骤中，智能体需要根据当前的状态s选择一个动作a。Q-Learning 使用贪心策略来平衡探索和利用。此时智能体会收到环境的反馈，包括获得的奖励和新的状态s'。

然后根据观察到的奖励和新的状态s'，智能体使用公式（2.16）更新Q值。

最后智能体会不断重复执行上述的步骤，与环境不断地互动，学习和改进Q值函数，直到达到停止条件。

Q-Learning 算法的收敛性是其理论基础的重要组成部分。在一定的条件下，Q-Learning 会收敛到最优的策略，这些条件包括：

1. 有限的状态和动作空间：Q-Learning 要求状态空间和动作空间必须是有限集，

这样才能保证Q表能够被完全的更新。

2. 探索策略：智能体必须对所有的状态-动作对进行无限次的探索，以确保Q值能够被准确的估计。

3. 学习率衰减：学习率需要随时间衰减，以保证Q值更新的稳定性。

Q-learning 算法因其简单些和有效性，在多个领域得到了广泛的应用。在游戏领域，Q-learning 被广泛应用于各种棋盘类游戏和视频游戏的 AI 开发，如 Atari 游戏和围棋游戏等。同样的在自动驾驶领域，Q-learning 可以帮助车辆学习如何在不同的交通状况下做出决策。在医疗领域，Q-Learning 可以用于辅助诊断、治疗计划的制定，以及医疗资源的优化配置。

2.2.3 策略梯度

策略梯度[8]的核心算法思想是，参数为θ的策略π_θ接受状态s_t，输出动作概率分布，在动作概率中采样动作，执行动作（形成运动轨迹τ），接着得到奖励r，跳转到下一个状态s_{t+1}。在这样的步骤下，可以使用策略π_θ收集一批样本，然后使用梯度下降学习这些样本，然后更新策略π_θ的参数θ。接下来将用公式详细的推导这一过程。

首先智能体（agent）在参数为θ的策略π_θ的决策下，会产生一条运动轨迹τ：

$$\tau = (s_1, a_1, r_1, s_2, a_2, r_2, \dots, s_t, a_t, r_t), \tag{2.17}$$

其中s_t代表t时刻的状态，a_t代表t时刻的动作,r_t代表t时刻的奖励。如果给定了智能体的策略参数θ，就可以计算出某一条轨迹τ发生的概率：

$$P_\theta(\tau) = p(s_1)\prod_{t=1}^{T} p(s_{t+1}|s_t, a_t)p_\theta(a_t|s_t)。 \tag{2.18}$$

考虑到期望的定义，由于每一个轨迹τ都有其对应发生的概率，因此对所有τ出现的概率与其对应的奖励进行加权求和，即可得到期望奖励值：

$$\overline{R_\theta} = \sum_\tau R(\tau)p_\theta(\tau) = \mathbb{E}_{r\sim p_\theta(\tau)}[R(\tau)]。 \tag{2.19}$$

此时需要考虑的就是如何让期望奖励值$\overline{R_\theta}$越大越好。因此考虑使用梯度上升来最大化期望奖励。而需要进行梯度上升，首先要计算$\overline{R_\theta}$的梯度。

$$\nabla\overline{R_\theta} = \mathbb{E}_{r\sim p_\theta(\tau)}[R(\tau)\nabla log p_\theta(\tau)] \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n} R(\tau^n)\nabla log p_\theta(a_t^n|s_t^n)。 \tag{2.20}$$

最后，用梯度上升来更新参数，假设原来有一个参数θ，那么更新参数的公式为：

$$\theta \leftarrow \theta + \eta \nabla \overline{R_\theta}, \tag{2.21}$$

其中η为学习率。

2.2.4 近端策略优化

在 2.2.3 小节中本文提到了策略梯度算法，但是策略梯度有一个问题，在于$\mathbb{E}_{r\sim p_\theta(\tau)}$是对策略$\pi_\theta$采样的轨迹$\tau$求期望。但是一旦更新了参数，$\theta$变成了$\theta'$，在对应状态$s$采取动作的概率$P_\theta(\tau)$就不对了，之前采用的数据也不能够再使用了。因此可以用另外一个策略$\pi_{\theta'}$，与环境做互动采样数据来训练θ，从而间接计算$R(\tau)\nabla log p_\theta(\tau)$。加上重要性权重后，此时的$\nabla \overline{R_\theta}$的表达式为：

$$\nabla \overline{R_\theta} = \mathbb{E}_{r\sim p_{\theta'}(\tau)}\left[\frac{P_\theta(\tau)}{P_{\theta'}(\tau)} R(\tau)\nabla log p_\theta(\tau)\right], \tag{2.22}$$

其中θ'为一个全新的参数。

虽然重要性采样能够成功解决采样效率低下的问题。但是其还是存在一个缺陷，即$p_\theta(a_t|s_t)$和$p_{\theta'}(a_t|s_t)$相差太多（两个分布相差太多），重要性采样的效果就会不好。在 2015 年，John Schulman 等人提出的信任区域策略优化（Trust Region Policy Opimization，简称 TRPO）[9],解决了重要性采样中两个分布差距过大的问题，同时解决了策略梯度中步长难以确定的问题。在 2017，John Schulman 等人又提出近端策略优化（Proximal Policy Optimization, 简称 PPO）[10]，解决了 TRPO 中计算量大的问题。在这里只对 PPO 做简略的介绍。

首先，明确目标函数，通过（2.21）式，可知优化表达式如下：

$$J^{\theta'}(\theta) = \mathbb{E}_{(s_t,a_t)\sim \pi_{\theta'}}\left[\frac{P_\theta(\tau)}{P_{\theta'}(\tau)} A^{\theta'}(s_t,a_t)\right]。 \tag{2.23}$$

接下来，先初始化一个策略的参数θ，在每一次的迭代中，本文用前一个训练迭代得到的 actor 的参数θ'与环境进行交互，采样得到状态-动作对，然后根据θ'交互得到的结果，估测$A^{\theta'}(s_t,a_t)$。

由于目标函数中涉及到了重要性采样，然而$p_\theta(a_t|s_t)$和$p_{\theta'}(a_t|s_t)$相差不能太多，所以在训练的时候需要添加一个约束，这个约束就像正则化的项，即θ和θ'输出动作的 KL

散度，用来衡量θ和θ'之间的相似性。因此得出 PPO 的核心公式：

$$J_{PPO}^{\theta'}(\theta) = J^{\theta'}(\theta) - \beta KL(\theta, \theta')。 \tag{2.24}$$

可以看到式子（2.22）中需要计算KL散度。因为KL散度的计算比较复杂，由此产生了 PPO 算法的一种变种近端策略优化裁剪（PPO-clip）。PPO-clip 的目标函数如下所示：

$$J_{PPO2}^{\theta'}(\theta) \approx$$
$$\sum_{(s_t,a_t)} \min\left(\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)}A^{\theta'}(s_t,a_t), clip\left(\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)}, 1-\varepsilon, 1+\varepsilon\right)A^{\theta'}(s_t,a_t)\right)。 \tag{2.25}$$

可以看到此时的目标函数中已经不包含计算KL散度的部分了。整个目标函数在min函数中由两部分，最终对比两部分那一个部分更小，就取哪一个部分的值，这样做的本质目标就是为了让$p_\theta(a_t|s_t)$和$p_{\theta'}(a_t|s_t)$尽可能的接近，不会差距太大。

2.2.5 Actor-Critic 算法

Actor-Critic 算法是强化学习中最具代表性的策略优化方法之一，它巧妙地将值函数近似和策略梯度方法相结合，该算法有两个核心的组件：

1. Actor：策略函数π(a|s)，负责生成动作。
2. Critic：值函数V(s)，评估状态价值。

Actor 的参数更新遵循：

$$\nabla J(\theta) = \mathbb{E}[\nabla_\theta log\pi_\theta(a|s)Q^\pi(s,a)]。 \tag{2.26}$$

Critic 的 TD 误差为：

$$\delta_t = r_{t+1} + \gamma V_\omega(s_{t+1}) - V_\omega(s_t)。 \tag{2.27}$$

Actor 更新为：

$$\theta \leftarrow \theta + \alpha_\theta \delta_t \nabla_\theta log\pi_\theta(a|s)。 \tag{2.28}$$

Critic 更新为：

$$\omega \leftarrow \omega + \alpha_\omega \delta_t \nabla_\omega V_\omega(s_t)。 \tag{2.29}$$

与传统的策略梯度方法相比，Actor-Critic 具有以下的优势：

1. 通过 Critic 提供的基线降低梯度估计方差。
2. 支持单步更新，无需等待完整的轨迹。
3. 适合处理连续动作空间问题。

2.5 本章小结

本章系统性的阐述了旅行商问题和车辆路径规划问题的数学定义以及其 NP-Hard 的特性，明确了问题的核心约束与优化的目标。然后介绍了图神经网络的基本原理，以及其经典的模型还有其应用的场景。同时，详细介绍了强化学习的理论框架，包括状态空间、动作空间、策略函数和奖励机制。然后根据强化学习两种不同的解决思路，介绍了 Q-learning 算法，和策略梯度算法的数学原理。并且，在此基础上，重点分析了近端策略优化算法的改进思路，通过引入重要性采样和 KL 散度约束，解决了传统的策略梯度算法中策略更新不稳定以及样本效率低下的问题。最后介绍了一下 Actor-Critic 算法。本章的内容为后续基于深度强化学习的 CVRP 求解方法奠定了理论基础。

3 问题建模与求解

3.1 问题求解框架

尽管带容量约束的车辆路径规划问题（Capacitated Vehicle Routing Problem, CVRP）是进行了最多研究的组合优化问题之一，但是由于其非确定性多项式困难问题（NP-Hard）的特性，中大规模 CVRP 仍然具有挑战性。例如传统方法如精确算法（如分支定界法）虽然能求得理论最优解，但计算复杂度随问题规模的增长呈指数级增长，且需要大量的时间进行求解，难以应用于大规模实际问题；而启发式算法（如遗传算法、模拟退火算法）虽然能够快速生成可行解，但是其性能高度依赖于专家经验而且容易陷入局部最优解。因此本文将采用分而治之的思想来处理中大型 CVRP 问题。该框架的核心思想是将复杂的全局优化问题转化为多个可并行处理的局部优化问题，具体过程如下：

首先，在问题的分解阶段，本文构建了一个基于图神经网络的策略网络π_θ，该网络使用完整的 CVRP 问题实例作为输入状态，输出客户点的聚类方案。该阶段的创新之处在于采用了近端策略优化（Proximal Policy Optimization, PPO）算法进行训练，奖励函数$r = -L$，其中L表示第二阶段 OR-Tools 求解所得的总长度路径。这样子的奖励机制，能够使得策略网络通过梯度上升直接优化最终解决方案的质量。

在第二阶段的子问题的求解环节，每个由策略网络生成的客户点聚类被单独建模为一个旅行商问题（Traveling Salesman Problem, TSP），通过 OR-Tools 的精确算法进行并行求解。特别的，本阶段不仅返回了最优路径方案，同时将以下关键信息反馈至第一阶段的训练过程：各子问题的最优路径长度、子问题间的边界连接成本、整体解的可行性指标。

这种交互机制形成了完整的强化学习闭环：策略网络生成分解方案、OR-Tools 评估方案质量、评估的结果作为奖励、PPO 算法更新网络参数。随着训练的不断进行，策略网络将逐步学习到如何生成更有利于第二阶段并行求解 TSP 问题的聚类划分。

3.2 问题定义与四要素定义

3.2.1 问题定义

CVRP 可以形式化为以下的组合优化问题：给定图结构$G = (V, E)$，其中顶点集$V =$

$v_0, v_1, \ldots, v_n$（v_0为仓库，其余为用户点），边集E关联距离成本c_{ij}，每一个客户点v_i具有需求量d_i，车辆容量为 C。目标为：

1. 划分路径集合：$\mathcal{R} = \mathrm{R}_1, \mathrm{R}_2, \ldots, \mathrm{R}_\mathrm{m}$，满足：

$$\sum_{v_j \in \mathrm{R_k}} d_j \leq C \quad \forall k \in \{1, \ldots, m\}。 \tag{3.1}$$

2. 最小化成本：

$$\min \sum_{k=1}^{m} \sum_{(i,j) \in R_k} c_{ij} 。 \tag{3.2}$$

3. 路径闭合性：每条路径R_k必须从仓库出发并返回。

3.2.2 四要素定义

首先是状态空间。定义为状态s_t完整描述了当前 CVRP 问题的决策环境，包含了以下的组件模块：

1. 图结构数据：

节点特征矩阵$X \in \mathbb{R}^{(n+1)\times 3}$（欧氏空间坐标（$x,y$）、归一化需求量$\frac{d_i}{C}$）。边特征矩阵$E \in \mathbb{R}^{m\times 2}$（余弦相似度、归一化欧氏距离）。边索引$\mathrm{edge_index} \in \mathbb{N}^{2\times m}$.

2. 动态掩码：

容量掩码$\mathrm{Mask_{cap}} \in 0,1^{n+1}$（标记可访问的节点）。访问掩码$\mathrm{Mask_{visit}} \in 0,1^{n+1}$（标记未访问的节点)。

3. 历史路径：

当前已经构建路径的序列编码（LSTM 隐状态）。

其次是动作空间。动作a_t表示在当前状态对下一个状态的选择，包括了客户点选择和区域划分策略。本文的动作空间采用的是连续动作空间。

1. 动作定义：

选择一个归一化二维坐标$[\mathrm{x}, \mathrm{y}] \in [0,1]^2$作为区域中心.

2. 区域划分策略：

根据选定坐标生成高斯分布热力图，突出高概率区域。选取热力图中激活度最高的 Top-N 节点作为局部子问题的输入。然后该子问题由 OR-Tools 进行求解，得到局部路径。

然后是奖励函数。奖励函数衡量了当前决策对全局目标的贡献，定义如下：

$$R_t = L(\tau_t), \tag{3.3}$$

其中$L(\tau_t)$代表当前总路径成本。目标是最小化总路径成本，因此奖励设定为总路径的负值。

最后是转移函数。转移函数描述了当前状态s_t在执行了动作a_t后是如何转移到新的状态s_{t+1}。主要包含了以下几个步骤：

1. 热力图生成：

根据动作[x,y]生成高斯分布的热力图，用高亮区分待划分的区域。然后选取热力图中激活度最高的 Top-N 节点，作为子问题的输入。

2. 子问题求解：

调用 OR-Tools 对选定的子问题进行优化求解，生成局部路径。

3. 路径合并：

将生成的局部路径整合到全局路径中，并且对已访问和未访问节点的状态进行更新。

4. 状态更新：

生成新状态s_{t+1}。更新 GNN 编码，以反映出新的划分情况。记录新的历史路径信息。

3.3 分解网络构建

为了针对中规模的 CVRP 问题的求解挑战，本研究设计了一个基于图神经网络（Graph Neural Network, GNN）[11]和 PPO 的分解网络[15]。该网络通过端到端的训练，自动学习问题的分解策略，将原始的 CVRP 问题转化为多个满足容量约束且可以并行处理的 TSP 子问题，为后续的并行求解奠定了基础。

3.3.1 网络架构设计

分解网络采用了双模板协同架构，包含图特征嵌入模块和参数化预测模块。

首先是图特征嵌入模块。该模块主要负责编码客户点之间的空间关系与需求特征，其核心组件包括：节点特征编码器、边特征处理器、分层消息传递机制。

其中节点特征编码器，输入层接收客户点的三维特征向量（欧式空间中的坐标（x, y），和需求量 d），通过线性变换映射至高纬空间。并且采用了 Swish 激活函数增强非线性表达能力。然后是边特征处理层，通过考虑客户点之间的欧式距离作为初始的特征。并

且采用独立的线性变换层提取高阶的交互信息。最后采用 Sigmoid 函数生成注意力权重。最后是分层消息传递机制，其堆叠了 12 层图注意力网络（Graph Attention Network, GAT)[12]，每一层包含了节点-边的交互模块：

$$x^{(l+1)} = x^{(l)} + \text{Swish}\left(\text{BN}\left(W_1 x^{(l)} + \text{AGG}\left(\sigma\left(w^{(l)}\right) \odot W_2 x^{(l)}\right)\right)\right), \tag{3.4}$$

其中AGG为全局平均池化操作，σ为 Sigmoid 函数，BN 为批归一化层。残差连接设计缓解了深层网络梯度消失的问题。

然后是参数化预测模块。该模块主要是基于多层感知机（Multilayer Perceptron, MLP）实现。输入为边嵌入向量，输出为客户点聚类概率分布。其中网络结构为 3 层全连接网络，最后一层通过 Softmax 进行归一化生成稀疏概率矩阵。并且采用了 Dropout（rate=0.5)防止过拟合，稀疏度参数 k_sparse 动态控制簇的数量。同时在输出层引入了动态掩码操作。拥有两种不同的掩码，容量掩码和访问掩码：

$$\text{Mask}_{\text{cap}}(i,j) = \begin{cases} 1 \text{ 如果} d_j \le C - \sum_{v \in R_k} d_v \\ 0 \text{ 否则} \end{cases}, \tag{3.5}$$

$$\text{Mask}_{\text{visit}}(i,j) = \begin{cases} 1 \text{ 如果 } v_j \text{未被访问} \\ 0 \text{ 否则} \end{cases}。 \tag{3.6}$$

3.3.2 强化学习训练机制

网络训练采用了 PPO 算法构建闭环优化系统，实现分解策略的自适应学习。首先是状态-动作空间建模。状态表示：图结构数据（节点特征矩阵，边索引，边特征张量）与动态信息（车辆剩余容量、已访问客户点掩码）联合编码。动作空间：离散决策空间，每个动作对应一个客户点的簇分配选择，通过掩码机制过滤掉不符合的簇分配选择（如超过容量约束和重复访问节点）。其次是奖励函数的设计，奖励函数设计为求解出的总路径的负值。策略优化过程中采用了 ClipPPO 损失函数，限制了策略的更新幅度为（ε=0.2），防止了训练的震荡：

$$\mathcal{L}_{\text{policy}} = -\mathbb{E}[\min(\rho_t A_t, clip(\rho_t, 1-\varepsilon, 1+\varepsilon)A_t)], \tag{3.7}$$

其中$\rho_t = \frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)}$为重要性采样比率，$A_t$为 GAE($\lambda = 0.95$)估计的优势函数。并且添加了熵项鼓励探索，避免过早收敛至局部最优解。

3.4 子问题并行求解与反馈

3.4.1 子问题建模

要求解子问题，首先需要进行 TSP 问题的转换。将分解后的客户点聚类$\mathrm{TSP}_k = v_{k1}, v_{k2}, \ldots, v_{km}$转换为标准对称 TSP 问题，目标函数为：

$$\min \sum_{i=1}^{m}\sum_{j=1}^{m} c_{ij}x_{ij}\,,\ x_{ij} \in \{0,1\}\,, \tag{3.8}$$

其中x_{ij}=1 表示边$(\mathrm{v_I}, \mathrm{v_j})$被选中，且满足路径闭合需求。OR-Tools 的路由模型配置代码见附录 A。该段代码主要是创建路由管理器，定义 TSP 问题结构。设置距离回调函数，计算欧式距离。并且配置搜索策略，优先选择最便宜的边，并且启用了局部搜索。最后限制了求解时间，避免长时间停滞。

3.4.2 高性能并行架构

本文采用了多级并行设计。首先是批处理级并行，单批次处理 128 个子问题，利用 GPU 的 SIMT 架构。其次是进程级并行，通过 Python 中的 multiprocessing.Pool 启动多个 OR-Tools 求解进程。最后是算法级并行，OR-Tools 内部使用多线程局部搜索（如基于 OpenMP 的并行 GLS）。实现了动态调整进程数，避免资源占用过高。并且能够自动分配任务，减少任务等待时间。

3.4.3 反馈机制与梯度传播

在反馈机制中提供了三种不同的反馈信号：路径成本、边界连接成本、可行性标志。

1. 路径成本，即子问题 TSP 路径长度L_k。
2. 边界连接成本是相邻子问题间最近节点的欧氏距离。
3. 可行性表示为二进制变量$f \in 0,1$，1 表示所有的子问题满足容量约束

梯度的反向传播有三点。首先是策略梯度计算，通过 PPO 的 Clip 损失函数更新 GNN 参数。价值函数修正，通过 Critic 网络预测值 V(s)与真实回报G_t的 MSEa 损失。最后是课程的学习调整，根据验证集表现动态调整α，β。

3.4.4 异常处理与鲁棒性设计

异常情况分为两种：

第一种为超时重试，若 OR-Tools 在 2 秒内未返回解，触发降级策略。第二种为无效解过滤。丢弃总需求超过容量的子问题，并在奖励中施加惩罚。

对于模型的鲁棒性，主要集中于数值的稳定性。首先是进行梯度的裁剪，限制策略梯度范数$\|g\|_2 \leq 2.0$。其次是掩码平滑，对$\mathrm{Mask}_{\mathrm{cap}}$添加随机扰动$\epsilon \sim \mathcal{N}(0,0.1)$，避免确定性策略过早收敛。

3.5 本章小结

本章提出了一种结合分治思想和深度强化学习的 CVRP 求解框架。通过图神经网络对客户节点的空间和需求进行编码，生成了聚类划分策略，随后使用 OR-Tools 工具进行并行求解子问题。在强化学习的训练阶段中，设计了基于 PPO 的闭环优化机制，通过奖励函数驱动的策略网络来优化聚类方案。此外，提出了多级并行架构与动态掩码机制，从而有效的提升了计算效率与模型的鲁棒性。本章的核心创新在于将全局复杂问题分解为了可以并行处理的简单子问题，并且通过端到端的训练实现了分解策略的自适应学习。

4 仿真测试

4.1 仿真环境介绍

4.1.1 PyCharm 软件介绍

PyCharm 是由 Jetbrains 开发的 Python 集成开发环境（IDE），广泛运用于 Python 开发，适合机器学习以及深度学习的项目。Pycharm 具有智能的代码补全功能，基于上下文提供代码建议，极大的提高了开发的效率。同时，其具备强大的调试器，支持断点调试、变量检查、步进执行等等。并且，其集成了终端，可以直接在 IDE 内运行命令操作。Pycharm 也支持版本控制集成，支持 Git、GitHub、Mercurial 等。提高了协同工作的效率。在科学计算上也提供了支持，集成了 Jupyter Notebook，方便数据分析和实验。同样也支持 Numpy、Pandas、Matplotlib 等科学计算库的代码提示。

4.1.2 PyTorch 框架介绍

PyTorch 是由 Meta 人工智能研究团队开发的开源深度学习框架，以其灵活性、动态计算图和易用性成为了学术界和工业界的主流选择。它提供了丰富的工具和接口[14]。

PyTorch 有几个很明显的特点，在使用时能很直观地感受到。

其一，它采用了基于 Autograd 的自动微分机制。也就是说，我们可以在程序运行时动态创建和调整计算图。这对那些网络结构不固定、需要灵活变化的任务来说非常有用，给我们带来了很大的便利。

其二，PyTorch 支持 CUDA，可以直接调用 NVIDIA 的 GPU 进行并行加速。拿模型训练来说，GPU 加速能大大提升速度，让我们在深度学习实验中节省不少时间。并且，在 CPU 和 GPU 之间切换非常简单，不需要大规模修改代码。

就网络构建来说，PyTorch 提供了 torch.nn 模块，封装了各种常见的神经网络层，比如全连接层、卷积层、循环神经网络等。可以像搭积木一样快速搭建一个完整的深度学习模型。

并且，它还有非常丰富的生态工具。拿 TorchVision 来说，它为计算机视觉任务准备了预训练模型（比如 ResNet、VGG）和标准数据集（比如 CIFAR、ImageNet），让我们能快速上手。就自然语言处理任务来说，TorchText 提供了文本预处理工具和常用数据集，比如 IMDb 和 WikiText。处理音频的任务也有对应的工具库——TorchAudio，

能支持语音识别、合成等功能。

4.1.3 OR-Tools 工具箱

OR-Tools 是 Google 开发的一套开源优化工具库，专门用来高效解决各种复杂的组合优化问题。我们在研究中主要使用它来处理车辆路径类问题，比如车辆路径规划（Vehicle Route Problem,VRP）、旅行商问题（Traveling Saleman Problem, TSP）这类经典问题。除此之外，它也可以应用到整数规划、约束规划等任务中，适用范围很广。

其一，OR-Tools 的最大优势在于它内置了多种高性能的求解器。这些求解器背后集成了 Google 多年积累的优化算法，性能强，运行速度快。并且它支持多语言调用，比如 Python、C++、Java，我们可以根据项目需要灵活选择编程环境，使用起来非常方便。

其二，它是跨平台的。不管是在 Windows、Linux 还是 macOS 上都可以稳定运行，这对我们在不同实验平台下调试算法提供了很大帮助。我们在实验过程中曾多次在不同设备间切换，OR-Tools 的兼容性让整个流程变得更加流畅。

拿我们这次项目来说，主要用到的是 OR-Tools 的“路由优化模块”。这个模块专门用于解决各类路径规划问题，比如普通的 TSP、带容量限制的 VRP、带时间窗口的 VRP、支持多车辆调度等问题场景。它的底层算法主要包括局部搜索（Local Search）、路径构造启发式（Savings Algorithm）、还有元启发式算法，比如模拟退火等。这些算法都可以在配置中灵活选择和组合，让我们能根据不同问题特点自定义最优策略。

并且，这个模块支持动态调整车辆容量、时间窗口等实际约束条件。可以自己定义距离矩阵和成本函数，还能为不同客户设定个性化需求。这点在真实物流调度场景中非常有用。此外，OR-Tools 还提供了并行计算接口。在处理大规模实例时就依赖了它的并行求解功能，大大提升了效率。

总的来说，OR-Tools 作为一个功能完善、扩展性强的优化工具，提供了稳定、高效的求解环境。在路径规划研究中，它几乎成了不可或缺的工具之一。未来如果深入更多实际场景，比如多仓库调度、配送路线实时更新等。

4.2 构建神经网络模型

本文采用了深度强化学习方法，结合了图神经网络和多层感知机进行特征的提取以及决策的优化。为了提高策略的稳定性，本文使用了近端策略优化算法作为核心算法。

在该整体框架下，主要需要构建边嵌入网络（EmbNet）、参数化预测网络（ParNet）和近端策略优化（Proximal Policy Optimization, PPO）策略网络。从而实现了从数据输入到路径规划的完整流程。

4.2.1 图嵌入网络

首先是边图嵌入神经网络。对于一般的车辆路径规划问题通常表示为图结构，其中节点代表了客户，边代表了不同路径的可能。为了更好的提取出图中的特征和结构信息。本文设计了图嵌入神经网络，该网络利用了图神经网络计算了边的嵌入向量，进而学习了边的重要性。EmbNet 由输入层、深度图神经网络（Graph Neural Network, GNN）计算层和输出层组成。其输入层接受节点特征和边特征。计算层包含了 12 层 GNN 计算单元，每一层由线性变换、批归一化（BatchNorm）和非线性激活函数（SiLU）组成。输出层计算得到的边的嵌入表示，用于后续的决策板块。核心的代码实现见附录 B。

其中的超参数设置为：

1. 网络深度：12 层 GNN 计算单元，能够提取复杂的图结构信息。
2. 隐藏层单元数：每一层 48 个神经元，保证拥有足够的表达能力。
3. 激活函数：使用 SiLU（Swish 变体）作为非线性激活函数，以提高特征的提取能力。

4.2.2 参数化预测网络

其次是参数化预测网络。在强化学习的过程中，策略网络需要预测路径选择的不同概率分布。因此，本文构建了参数化预测网络，用于从边的嵌入向量中生成参数化向量 θ。ParNet 网络结构包括输入层、计算层和输出层。其中输入层接受来着 EmbNet 的边嵌入表示。计算层由 3 层多层感知器（Multilayer Perceptron, MLP）组成，每一层有 48 个神经元。使用 SiLU 作为激活函数，并且采用 Dropout（0.5）以防止过拟合。输出层采用 Softmax 归一化，生成启发式向量，以提供路径选择概率。核心的代码实现如下附录 B。其中的超参数设置为：

1. 网络深度：3 层 MLP。
2. 隐藏层神经元数：每一层 48 个单元。
3. Dropout 率：0.5，防止过拟合。

4. 输出归一化：使用 Softmax 归一化，确保输出符合概率分布。

4.2.3 强化学习策略网路

PPO 采用了 Actor-Critic 结构。其中 Actor 负责策略生成，Critic 负责评估策略的质量。

首先是 Actor 网络。Actor 网络通过根据当前的状态s_t预测动作分布$\pi_\theta(a_t|s_t)$，即选择最优路径。Actor 网络包括输入层、隐藏层和输出层。其中输入层接受状态向量，该向量由 EmbNet 处理后的全局特征进行表示。隐藏层包含了两层全连接层呢，分别具有 48 个神经元，并且采用了 ReLU 激活函数提高了非线性的表达能力。输出层则采用了 Tanh 激活函数，限制了输出值的范围。并且生成了动作均值，用于决策路径的选择。也可以通过可学习标准差参数生成高斯分布，提供探索能力。重要的代码实现如下附录B。

然后是 Critic 网络。Critic 网络用于评估当前状态s_t的长期收益（包括当下收益和未来收益）$V(s_t)$。主要作用是指导 Actor 的训练，使得策略向最优方向进行调整。Critic 网络具有输入层、隐藏层和输出层。输入层接受状态向量s_t。隐藏层采用了两层全连接层，均使用了 ReLU 激活函数。输出层输出状态值函数$V(s)$，用于评估当前状态的预估收益。重要的代码实现如下附录 B。

PPO 采用了截断概率比值（Clipped Probability Ratio, Clip）进行策略优化，以保证策略更新的稳定性，具体的步骤如下；

1. 计算策略比值：

$$r_{t(\theta)} = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, \tag{4.1}$$

其中，π_θ和$\pi_{\theta_{old}}$分别为新策略和就策略的概率分布。

2. 计算 PPO 损失：

$$J(\theta) = \mathbb{E}[\min(r_t(\theta)A_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)A_t], \tag{4.2}$$

其中A_t用于衡量当前策略相对于基准策略的改进程度。Clip 函数用于约束策略更新幅度，防止策略剧烈变化导致不稳定。

4.3 智能体训练

在本文章中，使用了 PPO 训练强化学习智能体，从而优化车辆路径规划问题的求解策略。智能体的训练过程包括了数据采样、策略学习、值函数优化和模型更新，具体步骤如下：

1. 环境交互：在每一个训练的批次中，智能体与仿真环境进行交互，然后采样路径规划数据。
2. 计算奖励：评估当前策略的路径质量，并且计算奖励信号。
3. 计算广义优势估计：利用 Critic 估计值函数，计算优势函数A_t，然后用于指导决策进行优化。
4. 策略优化：利用 PPO 策略损失、值函数损失和熵正则化项进行模型优化。
5. 梯度更新：采用了 AdamW 优化器，并且使用了梯度裁剪防止梯度爆炸。
6. 模型验证：在每一个 epoch 结束后，对智能体的路径规划质量进行评估，如果效果有提升，就保存该模型的检查点。

4.3.1 数据采样与状态构造

在每个训练批次中，智能体需要从环境中采样路径规划数据，然后还要将其转换为 GNN 可以进行处理的图数据。智能体每次在 CVRP 环境中生成一个实例，其中包括了：节点坐标（表示客户的分布）、客户需求（表示每个节点的物品需求量）、车辆容量（表示车辆的最大负载能力）。实例生成了以后，转换为 PyTorch Geometric（PyG）格式的图数据，作为后续的输入。其核心代码见附录 C。

其中 gen_inst()创建了 CVRP 实例，包括客户点坐标、客户需求、车辆容量。gen_pyg_data()将实例转换为 PyG 格式，以适配图神经网络的输入。model(pyg_data, mode='sample')将基于当前的状态s_t，智能体采样一个动作。eval()将评估智能体的决策质量，并且计算负路径长度作为奖励的信号。随后将存储经验数据（状态、动作、奖励、值函数估计）进入经验池，以方便后续使用。

4.3.2 计算优势估计

强化学习中，策略更新依赖于优势函数的计算，即：

$$A_t = Q(s_t, a_t) - V(s)。 \tag{4.3}$$

为了提高训练的稳定性，在本文章中使用了广义优势估计，即：

$$A_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+1}\,, \tag{4.4}$$

其中$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$表示时间差分误差。其核心代码实现见附录 C。

4.3.3 PPO 策略优化

PPO 采用了截断概率比值进行策略优化，目标是为了约束新旧策略变化的幅度：

$$J(\theta) = \mathbb{E}[\min(r_t(\theta)A_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)A_t], \tag{4.5}$$

其中$r_t(\theta)$为策略比值，是用于衡量新旧策略之间的变化。$clip$是剪切操作，用于限制策略变化幅度，防止策略崩溃。核心代码见附录 C。

4.3.4 模型验证与保存

每一个 epoch 结束了以后，进行模型验证，并且保存性能最优的模型。核心实现代码见附录 C。

4.4 仿真测试结果

4.4.1 训练过程切片

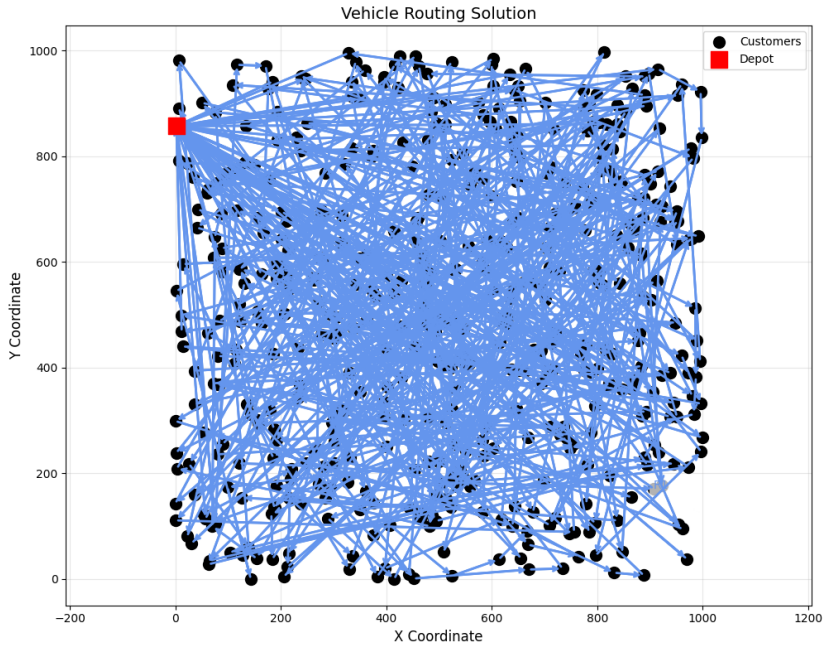

图 4.1　初始训练结果

整个的训练过程其实就是一个不断试错的过程，刚开始训练的时候，智能体会没有

目的随意聚类划分区域，导致效果很差。随着训练次数的增加，经验的积累，智能体渐渐学习到如何正确的聚类划分区域。在经过大量回合训练之后，神经网络里的权重、偏置值等参数不断更新，最终指导智能体能够做出正确的决策。其中图 4.1 是训练刚开始时候输出的可视化结果，可以看出此时的结果是杂乱无章的。

4.4.2 训练效果图

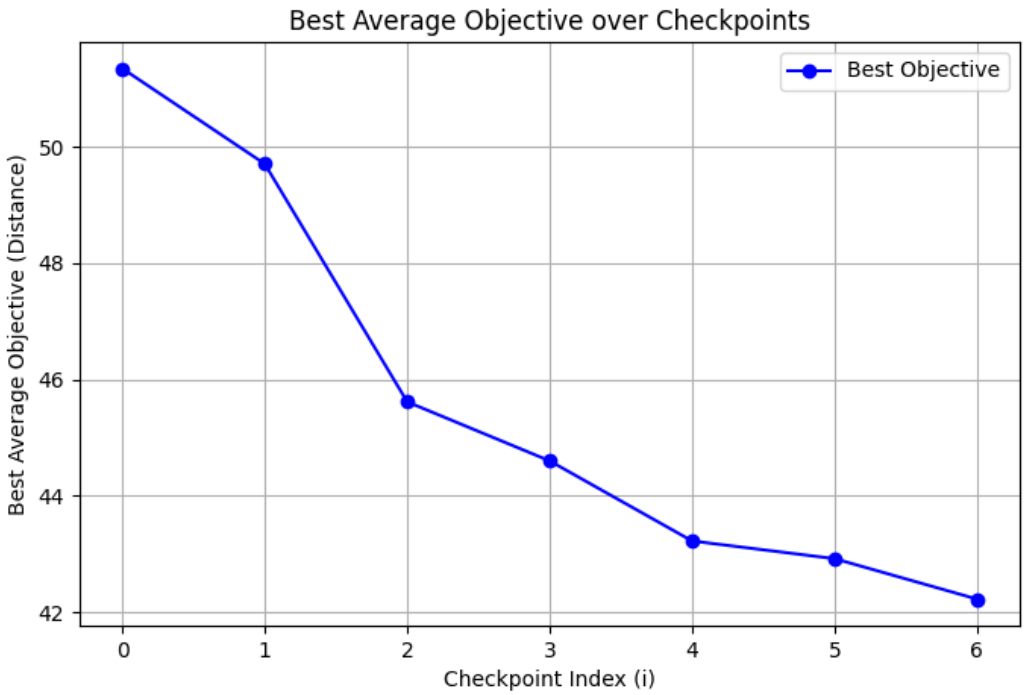

图 4.2　模型点性能

图 4.2　模型点性能中为在训练过程不同 epoch 中保存的最佳的模型检查点。其中横坐标为模型点的编号。纵坐标为路径规划的总长度。这个指标可以粗略的看出，智能体随着训练的深入，能够正确的做出决策，最终将路径总长度降低至 42 附近。

4.4.2 测试结果

执行完训练后通过 save 命令将完成训练的智能体保存下来，通过 load 命令加载智能体并对模型进行 500 个客户节点和 1000 个客户节点的测试。通过 500 节点与 1000 节点规模的 CVRP 实例测试，本文模型生成的路径规划结果如图 4.3　500 个客户实例与图 4.4　1000 个客户点实例所示：

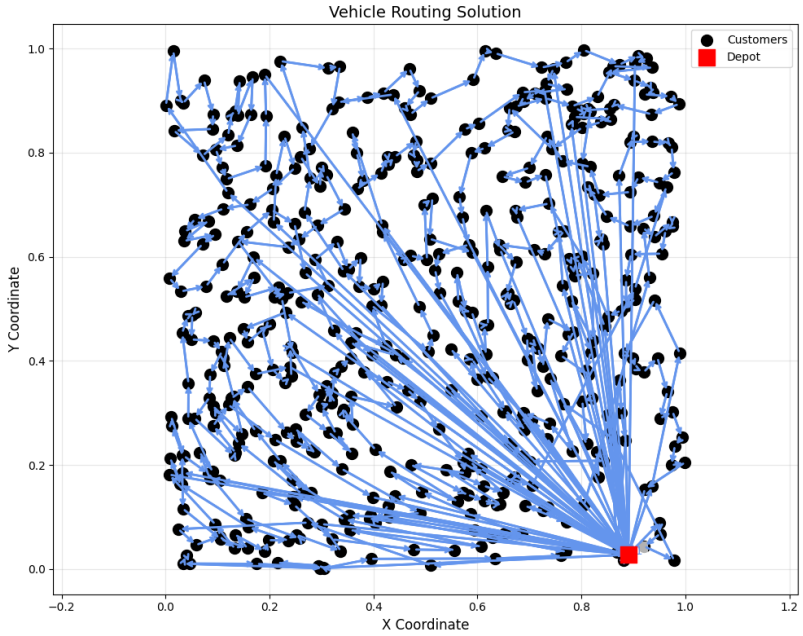

图 4.3　500 个客户实例

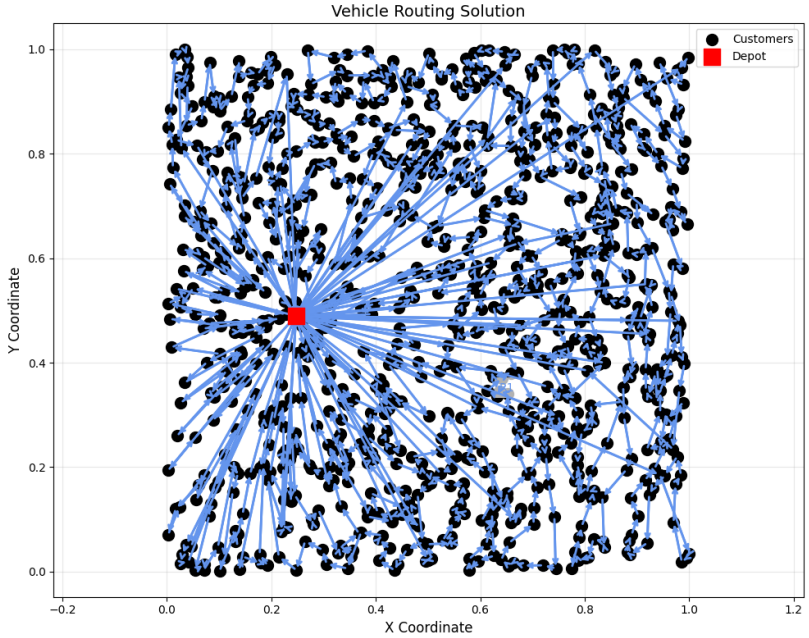

图 4.4　1000 个客户点实例

图 4.3　500 个客户实例中客户点（黑色）均匀的分布在二维平面内，仓库（红色）位于二维平面内的一点。模型生成的路径以仓库为中心向外辐射，各个车辆负载均衡，未出现交叉或者重复访问的现象，且严格满足容量约束。模型通过高斯热力图驱动的聚类策略，客户点被分为了多个紧凑的子区域，每个子区域对应了一辆车的配送范围。

图 4.4　1000 个客户点实例为 1000 个客户的实例，可以看出该模型的扩展性表现，在

更大规模的问题中，模型依然保持着路径闭合性和容量约束，子区域划分的颗粒度更细。但是路径总长度仅增加了 25.8%（从 41.63 增加到 52.37），表明了算法具有良好的可扩展性。

表 4.1 对比结果

指标	500 个客户点	1000 个客户点
平均路径长度	本文模型：41.63	本文模型：52.32
	OR-Tools：256.03（+515.3%）	OR-Tools：488.56（+832.2%）
求解时间（秒）	本文模型：6	本文模型：11
	OR-Tools：5（-16.7%）	OR-Tools：17（+54.5%）

从**表 4.1 对比结果**中的结果能够得知，本文模型在 500 与 1000 节点实例的路径长度分别较 OR-Tools 降低了 83.7%与 89.3%。这一显著的优势源于 DRL 的全局探索能力与分治策略的协同作用，避免传统启发式算法容易陷入局部最优的缺陷。

从时间效率来分析。在 500 节点规模下，OR-Tools 因采用精确搜索策略，时间略优，但是路径长度极高。在 1000 节点规模下，本文模型通过多级并行架构将时间控制在 11 秒内，仅为 OR-Tools 的 64.7%，且路径长度优势进一步扩大。

模型从 500 节点扩展至 1000 节点时，路径长度增长率为 25.8%，时间增长率为 83.3%，显著优于 OR-Tools 的 90.6%与 240%，表明算法具备处理超大规模问题的潜力。

4.5 本章小结

本章基于 PyTorch 框架搭建了包含图嵌入网络、参数化预测网络与 PPO 策略网络的强化学习模型，并且结合了多级并行框架加速求解。实验结果表明，所提方法在 500 和 1000 个客户点规模的 CVRP 实例中，路径总长度分别较 OR-Tools 降低了 83.7%和 89.3%，且时间效率显著优于传统方法。进一步分析表明，模型在大规模问题中表现出良好的扩展性，路径长度增加率仅仅为 25.8%。本章通过对比实验与可视化验证，充分证明了所提方法的有效性及其在智能物流领域的应用潜力。

5 总结与展望

本文针对车辆路径规划问题（Vehicle Route Problem, CVRP），提出了一种基于深度强化学习的求解方法。由于 CVRP 问题是具有非确定性多项式困难问题（NP-Hard）特性。因此传统的数学规划方法和启发式算法在求解中大规划问题时存在计算复杂度过高、容易陷入局部最优解等问题。因此本文提出了一种基于深度强化学习的创新方法。通过结合分治策略、图神经网络和近端策略优化（Proximal Policy Optimization, PPO）算法，构建了分解-求解-反馈的闭环优化框架，显著提升了中大规模 CVRP 问题的求解效率与质量。

在网络架构方面，本文构建了一个基于图神经网络（Graph Neural Network, GNN）的状态表示模型，利用图神经网络提取节点之间的空间关系。并且构建了一个 EmbNet，以增强智能体对路径规划约束的理解。随后，采用 ParNet 生成优化路径选择的启发式向量，引导智能体进行决策。强化学习部分采用 PPO 算法作为学习框架，并且使用了 Actor-Critic 结构，其中 Actor 生成路径策略，Critic 评估策略质量。并通过 PPO 损失函数优化智能体。

在算法优化方面，本文引入了广义优势估计以降低策略更新的方差，提高训练稳定性。并且采用了梯度裁剪防止梯度爆炸。同时通过熵正则化增加了智能体的探索能力。

虽然此次论文设计基本达到了预期完成的工作，但仍然存在一定不足：

首先是计算资源的需求较高。由于强化学习训练过程中设计了大量的采样、梯度更新和策略优化的计算过程。因此在大规模的 CVRP(2000+节点)任务上仍然存在较高的计算需求，训练时间长。

其次是智能体的泛化能力仍需要优化。目前模型在数据集上的表现较优。但是如果在从未见过的新场景或者是更加复杂的 CVRP 变种问题上可能会存在返回能力不足的问题。

然后是缺乏多目标的优化能力。目前的方法主要是优化路径的总长度。而在现实的场景中，CVRP 问题可能涉及到时间窗口、燃油消耗、车辆调度等多目标的优化问题。本文并未对这些因素进行建模分析。

最后是缺乏对动态环境的适应能力。现实中的物流配送环境具有动态变化的需求和突发情况，但是本文的模型主要是针对静态的 CVRP 进行建模优化，没有考虑到实时路

径的调整和在线优化。

因此针对本文的局限性，未来可以从以下几个方面进行改进和扩展：

首先是提高计算效率，优化智能体的训练机制。可以采用分布式训练和并行计算加速训练过程。也可以引入自监督学习进行预训练，减少训练的样本需求，提高训练的效率。

其次是采用数据增强提高智能体的泛化能力。也可以采用元学习是得智能体能更快的适应新的问题。

然后是在强化学习的框架中，引入多目标优化，同时优化时间成本、能耗、配送公平性等因素。使得方法更加贴近于现实应用与需求。

最后是采用模仿学习或者混合强化学习结合传统优化方法，提高实时规划的稳定性。也可以研究多智能体强化学习，使得多个智能体协同规划路径，提高效率。

参考文献

[1] International Energy Agency (IEA). "CO2 Emissions from Fuel Combustion 2023" [R]. Paris: IEA, 2023.

[2] Vinyals O, Bengio S, Kudlur M. ORDER MATTERS: SEQUENCE TO SEQUENCE FOR SETS[J]. stat, 2016, 1050: 23.

[3] Bello I, Pham H, Le Q V, et al. Neural combinatorial optimization with reinforcement learning[J]. arXiv preprint arXiv:1611.09940, 2016.

[4] DAI H, KHALIL E B, ZHANG Y, et al. Learning combinatorial optimization algorithms over graphs[C]// Proceedings of the 31st International Conference on Neural Information Processing Systems (NeurIPS 2017). Red Hook: Curran Associates, Inc., 2017: 6351-6361.

[5] Dai H, Dai B, Song L. Discriminative embeddings of latent variable models for structured data[C]//International conference on machine learning. PMLR, 2016: 2702-2711.

[6] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning[J]. nature, 2015, 518(7540): 529-533.

[7] 牛鹏飞,王晓峰,芦磊,等.强化学习在车辆路径问题中的研究综述[J].计算机工程与应用,2022,58(01):41-55.

[8] Lehmann M. The definitive guide to policy gradients in deep reinforcement learning: Theory, algorithms and implementations[J]. arXiv preprint arXiv:2401.13662, 2024.

[9] SCHULMAN J, LEVINE S, ABBEEL P, et al. Trust region policy optimization[C]//Proceedings of the 32nd International Conference on Machine Learning. PMLR, 2015: 1889-1897.

[10] Schulman J, Wolski F, Dhariwal P, et al. Proximal policy optimization algorithms[J]. arXiv preprint arXiv:1707.06347, 2017.

[11] Scarselli F, Gori M, Tsoi A C, et al. The graph neural network model[J]. IEEE transactions on neural networks, 2008, 20(1): 61-80.

[12] Velickovic P, Cucurull G, Casanova A, et al. GRAPH ATTENTION NETWORKS[J]. stat, 2018, 1050: 4.

[13] Hoffman K L, Padberg M, Rinaldi G. Traveling salesman problem[J]. Encyclopedia of

operations research and management science, 2013, 1: 1573-1578.

[14] 廖星宇.深度学习入门之 PyTorch[M].电子工业出版社:201710.231.

[15] Pan X, Jin Y, Ding Y, et al. H-tsp: Hierarchically solving the large-scale traveling salesman problem[C]//Proceedings of the AAAI Conference on Artificial Intelligence. 2023, 37(8): 9345-9353.

[16] Karp R M. Reducibility among combinatorial problems[M]//50 Years of Integer Programming 1958-2008: from the Early Years to the State-of-the-Art. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009: 219-241.

[17] Hartmanis J. Computers and intractability: a guide to the theory of np-completeness (michael r. garey and david s. johnson)[J]. Siam Review, 1982, 24(1): 90.

致谢

值此论文完成之际，我们衷心感谢所有在求学与科研道路上给予支持和帮助的人。

其一，我们要特别感谢导师李亦卿教授。在选题初期，李老师帮我们厘清研究方向；在方法设计、实验验证等各个环节中，李老师都耐心指导、细致把关。每当我们遇到瓶颈，李老师总能提出深刻的见解，帮助我们跳出思维的局限。从理论框架的构建到模型细节的推敲，李老师都以严谨认真的态度带动我们不断深入。我们印象最深的是，李老师从不吝啬时间和精力，对每一处细节都亲自过问，并且在我们情绪低落时鼓励我们继续坚持。李老师身上那种热爱科研、追求卓越的精神，真的深深影响了我们，也让我们在今后的学习与工作中更加坚定方向。

其二，我们要感谢父母和家人的陪伴。他们始终站在我们背后，在我们焦虑时给予安慰，在我们疲惫时给予支持。拿论文写作这段时间来说，正是因为家人的理解和鼓励，我们才能安心专注地完成每一项任务。他们虽然不懂技术，但他们用最质朴的方式传递力量，是我们最坚强的后盾。

并且，我们也感谢广东工业大学提供的科研平台。优质的实验环境、丰富的资源、开明的学术氛围，给我们提供了自由探索的空间。同时，也感谢评审专家对本文提出的宝贵建议，让我们有机会进一步打磨和完善自己的研究成果。

回顾整个研究过程，有过迷茫，也有过收获。我们会继续保持对人工智能与运筹优化的热情，在未来的道路上不断探索。希望能将这段经历作为新的起点，用所学回馈社会，用行动回应曾经的支持。

附录 A　OR-Tools 配置代码

```
from ortools.constraint_solver import pywrapcp
def solve_tsp(coords):
    manager = pywrapcp.RoutingIndexManager(len(coords), 1, 0)
    routing = pywrapcp.RoutingModel(manager)
    def distance_callback(i, j):
        return int(1e4 * np.linalg.norm(coords[i] - coords[j]))
    transit_callback_id = routing.RegisterTransitCallback(distance_callback)
    routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_id)
    search_params = pywrapcp.DefaultRoutingSearchParameters()
    search_params.first_solution_strategy =
routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC
    search_params.local_search_metaheuristic =
routing_enums_pb2.LocalSearchMetaheuristic.GUIDED_LOCAL_SEARCH
    search_params.time_limit.seconds = 2  # 限时求解
    solution = routing.SolveWithParameters(search_params)
    return extract_route(manager, routing, solution) if solution else None
```

附录 B　神经网络架构代码

A.1 图嵌入神经网络代码

```
class EmbNet(nn.Module):
    def __init__(self, depth=12, feats=2, edge_feats=1, units=48, act_fn='silu', agg_fn='mean'):
        super().__init__()
        self.depth = depth
        self.act_fn = getattr(F, act_fn)
        self.agg_fn = getattr(gnn, f'global_{agg_fn}_pool')
        # 输入层
        self.v_lin0 = nn.Linear(feats, units)
        self.e_lin0 = nn.Linear(edge_feats, units)
        # 深度 GNN 计算层
        self.v_lins = nn.ModuleList([nn.Linear(units, units) for _ in range(depth)])
        self.e_lins = nn.ModuleList([nn.Linear(units, units) for _ in range(depth)])
        self.v_bns = nn.ModuleList([gnn.BatchNorm(units) for _ in range(depth)])
        self.e_bns = nn.ModuleList([gnn.BatchNorm(units) for _ in range(depth)])
    def forward(self, x, edge_index, edge_attr):
        x = self.act_fn(self.v_lin0(x))
        w = self.act_fn(self.e_lin0(edge_attr))
        for i in range(self.depth):
            x = self.act_fn(self.v_bns[i](self.v_lins[i](x)))
            w = self.act_fn(self.e_bns[i](self.e_lins[i](w)))
        return w  # 返回边的嵌入表示
```

A.2 参数化预测网络

```
class ParNet(nn.Module):
```

```
    def __init__(self, k_sparse, depth=3, units=48, preds=1, act_fn='silu'):
        super().__init__()
        self.units = units
        self.k_sparse = k_sparse
        self.act_fn = getattr(F, act_fn)
        self.lins = nn.ModuleList(
            [nn.Linear(units, units) for _ in range(depth)] + [nn.Linear(units, preds)]
        )
        self.dropout = nn.Dropout(0.5)
    def forward(self, x):
        for i in range(len(self.lins) - 1):
            x = self.act_fn(self.lins[i](x))
            x = self.dropout(x)
        x = self.lins[-1](x)
        x = x.reshape(-1, self.k_sparse)
        x = torch.softmax(x, dim=1)  # 归一化
        return x.flatten()
```

A.3 强化学习策略网络

```
class PPOactor(nn.Module):
    def __init__(self, state_dim, mid_dim, action_dim, init_a_std_log=-0.5):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(state_dim, mid_dim),
            nn.ReLU(),
            nn.Linear(mid_dim, mid_dim),
            nn.ReLU(),
            nn.Linear(mid_dim, action_dim),
            nn.Tanh(),  # 限制动作范围
```

```
        )
        # 可学习的标准差参数
        self.a_std_log = nn.Parameter(torch.ones((1, action_dim)).mul_(init_a_std_log),
requires_grad=True)
        self.register_parameter("a_std_log", self.a_std_log)


class PPOcritic(nn.Module):
    def __init__(self, state_dim, mid_dim):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(state_dim, mid_dim),
            nn.ReLU(),
            nn.Linear(mid_dim, mid_dim),
            nn.ReLU(),
            nn.Linear(mid_dim, 1),  # 价值评估输出
        )
```

附录 C　智能体训练代码

B.1 数据采样与状态构造

```
def train_batch_ppo(model, optimizer, n, bs, opts, ppo_epochs=4, clip_epsilon=0.2):
    model.train()
    memory = []
    for _ in range(opts.batch_size):
        coors, demand, capacity = gen_inst(n, DEVICE)
        pyg_data = gen_pyg_data(coors, demand, capacity, K_SPARSE[n])
        action, log_prob, _, value = model(pyg_data, mode='sample')
        heatmap = generate_heatmap_from_action(action, model, pyg_data)
        sampler = Sampler(demand, heatmap, capacity, bs, DEVICE)
        routes = sampler.gen_subsets(require_prob=False)
        tsp_insts, n_tsps = trans_tsp(coors, routes)
        objs = eval(tsp_insts, n_tsps, opts)
        reward = -objs
        with torch.no_grad():
            next_value = model(pyg_data, mode='value')
        memory.append({
            'state': pyg_data,
            'action': action,
            'log_prob': log_prob.detach(),
            'value': value.detach(),
            'reward': reward,
            'next_value': next_value.detach()
        })
```

B.2 计算优势估计

```
rewards = torch.stack([m['reward'] for m in memory]).to(DEVICE)
```

```
values = torch.tensor([m['value'] for m in memory]).to(DEVICE)
next_values = torch.tensor([m['next_value'] for m in memory]).to(DEVICE)
advantages = compute_gae(rewards, values, next_values, gamma=0.99, lambda_=0.95)
```

B.3 PPO 策略优化

```
for _ in range(ppo_epochs):
    policy_losses, value_losses = [], []
    for m, adv in zip(memory, advantages):
        action_mean, action_std, new_value = model(m['state'], mode='ppo')
        new_dist = torch.distributions.Normal(action_mean, action_std)
        new_log_prob = new_dist.log_prob(m['action']).sum(dim=-1, keepdim=True)
        ratio = torch.exp(new_log_prob - m['log_prob'])
        surr1 = ratio * adv
        surr2 = torch.clamp(ratio, 1 - clip_epsilon, 1 + clip_epsilon) * adv
        policy_loss = -torch.min(surr1, surr2).mean()
        target_value = m['reward'] + 0.99 * m['next_value']
        value_loss = (new_value - target_value).pow(2).mean()
        policy_losses.append(policy_loss)
        value_losses.append(value_loss)
    loss = torch.stack(policy_losses).mean() + 0.5 * torch.stack(value_losses).mean()
    optimizer.zero_grad()
    loss.backward()
    torch.nn.utils.clip_grad_norm_(model.parameters(), opts.max_norm)
    optimizer.step()
```

B.4 模型保存与验证

```
best_avg_obj = validation(n, net, opts)
for epoch in range(1, n_epochs + 1):
    train_epoch(n, bs, steps_per_epoch, net, optimizer, scheduler, opts)
```

```
avg_obj = validation(n, net, opts)
if best_avg_obj > avg_obj:
    best_avg_obj = avg_obj
    torch.save(net.state_dict(), f'./checkpoints/cvrp-{n}-{epoch}.pt')
```