面向异构多背包问题的深度强化学习算法

李斌[1,2]，郭毅[2,3]

（1. 福建理工大学机械与汽车工程学院，福建 福州 350118；
2. 福建理工大学福建省大数据挖掘与应用技术重点实验室，福建 福州 350118；
3. 福建理工大学计算机科学与数学学院，福建 福州 350118）

摘　要：从传统多背包问题（KP）与典型物流系统运作场景出发，抽象出异构多背包问题（HMKP），并制定改进深度确定性策略梯度（DDPG）算法对 HMKP 进行研究和求解。针对 DDPG 算法在解决 0-1 KP 时容易陷入局部最优的缺点，采用动态随机机制（DRM）和动态惩罚机制（DPM）对 DDPG 算法进行改进，并嵌入改进 Transformer 模块来优化算法，提出基于改进 Transformer 模块的动态深度确定性策略梯度（TDP-DDPG）算法，并加入禁忌表防止重复搜索。TDP-DDPG 算法在多个实验算例中展现了高效的搜索能力，在由低到高维度的测试集 1、2 以及更高维度的测试集 3 中所有 39 个算例都能找到最优值，在大规模测试集 4 的 6 个算例中有 3 个能找到最优值。实验表明，TDP-DDPG 算法在融入改进策略后具备更强的寻优能力。在此基础上，设计基于 TDP-DDPG 算法的 BPD-DDPG 算法来解决复杂度更高的 HMKP，且分别在多个经典 0-1 KP 算例组合而成的高维度算例中进行分析评估。结果显示 BPD-DDPG 算法与商业求解器 Gurobi 相比虽求解时间长，但在 3 个低规模算例中求解准确率比 Gurobi 高。BPD-DDPG 算法能在可接受时间范围内以低计算代价高效解决高维度、大规模的 HMKP。

关键词：深度强化学习；0-1 背包问题；异构多背包问题；Transformer 模块；动态惩罚机制；禁忌表
中图分类号：TP18　　**文献标志码：**A　　**DOI：**10.19678/j.issn.1000-3428.0070166

Deep Reinforcement Learning Algorithms for Heterogeneous Multiple Knapsack Problems

LI Bin[1,2], GUO Yi[2,3]

(1. School of Mechanical and Automotive Engineering, Fujian University of Technology, Fuzhou 350118, Fujian, China;
2. Fujian Provincial Key Laboratory of Big Data Mining and Applications, Fujian University of Technology, Fuzhou 350118, Fujian, China;
3. School of Computer Science and Mathematics, Fujian University of Technology, Fuzhou 350118, Fujian, China)

【Abstract】 By focusing on the traditional multi-Knapsack Problem (KP) in typical logistics system operations, this study abstracts a Heterogeneous Multiple Knapsack Problem (HMKP) and formulates an improved Deep Deterministic Policy Gradient (DDPG) algorithm to solve it. The DDPG algorithm tends to fall into a local optimum when solving the 0-1 KP. To address this issue, a Dynamic Randomization Mechanism (DRM) and Dynamic Penalty Mechanism (DPM) are adopted and embedded with an improved Transformer module to optimize the algorithm. Then, a Dynamic DDPG (TDP-DDPG) algorithm is proposed based on the improved Transformer module. First, a tabu list is added to prevent repeated searches. The TDP-DDPG algorithm demonstrates efficient search capability across several experimental algorithms, finding the ideal optimum in 39 classical algorithms in test sets 1 and 2 from low to high dimensionality, as well as in higher dimensionality test set 3 and in three out of six algorithms in large-scale test set 4. Experiments show that the TDP-DDPG algorithm has stronger optimization seeking ability after incorporating the improved strategy. Next, the BPD-DDPG algorithm is designed based on the TDP-DDPG algorithm to solve the HMKP with higher complexity and is analyzed and evaluated in high-dimensional arithmetic by combining several classical 0-1 KP examples. The results show that the BPD-DDPG algorithm is more accurate than Gurobi in three low-scale cases; however, the solution time is longer. The BPD-DDPG algorithm can efficiently solve high-dimension, large-scale HMKP at a low computational cost within an acceptable time.

【Key words】 Deep Reinforcement Learning (DRL); 0-1 Knapsack Problem (KP); Heterogeneous Multiple Knapsack Problem (HMKP); Transformer module; dynamic penalty mechanism; tabu list

---

基金项目：教育部人文社会科学研究规划基金（19YJA630031）。

作者简介：李斌（CCF 高级会员），男，教授、博士后，主研方向为机器学习、群集智能、智慧港航；郭毅，硕士研究生。

收稿日期：2024-07-23　　**修回日期：**2024-09-13　　**E-mail：**whutmse2007_lb@126.com

0 引言

背包问题(KP)[1]是组合优化领域中的经典的NP-hard 问题。该问题可描述为在有限的背包容量内,使放入背包的物品总价值最大。现实生活中,物品总价值还可以被抽象为各种约束条件,并广泛应用于各个领域。如物流领域中的货物装载问题,物流公司需要根据车辆的容量,以及货物的体积、重量等因素来优化装载,从而使运输成本最小。此外,在金融领域中,为了在特定的风险下获取最高的收益,需要根据所持有的资金进行投资分类上的组合优化。背包问题由于其应用的广泛性、理论的挑战性,一直以来都是备受关注的热点问题之一。

KP 的变体主要包括 0-1 KP、多重 KP、多维 KP(MKP)、分数 KP 等。其中 0-1 KP 为最基础的 KP,其他类型的 KP 也可以转化为 0-1 KP 进行求解。目前求解 0-1 KP 通常使用的是精确算法和启发式算法。精确算法包括贪心算法[2]、动态规划法[3]、回溯法[4]等,启发式算法包括人工蜂群算法[5-6]、鲸鱼算法[7-8]、蚁群算法[9-10]、狼群算法[11-12]、狮群算法[13-14]等。精确算法一般只能在很小规模的算例中找到最优解,随着问题规模的增大,其极大的计算量会导致求解时间延长甚至出现在一定时间内无解的情况,并且精确算法一般对求解问题的形式要求比较严格,这限制了其应用范围。启发式算法相较于精确算法在性能上有所提升,但在问题维度提升和规模扩大的情况下,其性能表现往往受到影响。此外,启发式算法由于依赖固定的规则,灵活性较低,难以适应复杂 KP 的变种。

近年来,深度强化学习(DRL)取得了显著的发展。作为人工智能领域的重要分支,深度学习表现出来的强大表述能力和强化学习较强的决策能力使得 DRL 算法在复杂的求解环境中展现出优异的性能。DRL 作为深度学习和强化学习的结合,在广泛的应用场景和不同的模型中表现出很强的自适应学习能力、全局优化能力和泛化性能。随之引起了研究者利用 DRL 对组合优化问题的求解。WANG 等[15]提出了生成逆强化学习,并且通过结合生成对抗网络和 DRL 来提高算法的泛化能力,解决了稀疏奖励问题。BELLO 等[16]专注于旅行商问题(TSP),通过训练一个递归神经网络来预测不同城市排列分布。同时,此方法也为 KP 提供了相似的求解思路。KOOL 等[17]提出了一个优于 Pointer Network 的基于注意力层的模型,并以此来改进有关 TSP 的启发式算法。HU 等[18]提出了一种双目标双向的 DRL 算法,以减少延误为目的确定 TSP 的解决方案。WANG 等[19]提出一种求解大规模 TSP 的动态图卷积 Conv-LSTM 模型,使用动态的编码-解码方式和长短期记忆网络(LSTM)使模型能动态捕获图的拓扑结构。WANG 等[20]提出一个两阶段多智能体强化学习模型来解决车辆路径问题(VRP)。KHALIL 等[21]提出将深度图嵌入与强化学习结合,解决了基于图上的各种优化问题。SUI 等[22]提出一种 Neural-3-opt 的迭代改进方法,通过强化学习自动学习有效的 3-opt 来解决 TSP。

此外,有关使用强化学习解决组合优化问题中的 NP-hard 问题[23-24]的研究多集中于 TSP[25]和 VRP[26]这两种问题,对于 KP 的研究相对来说比较匮乏。近年来,对 KP 的研究更多使用的是启发式算法,然而在高维度、大规模算例的实验中其性能表现不稳定,求解成功率较差。为解决上述问题,本文基于 Transformer 模块[27-28]提出一种改进的深度确定性策略梯度(DDPG)算法,称之为 TDP-DDPG 算法。相比之下,DRL 算法在高维度、大规模问题的求解中展现出显著的精度和稳定性优势,并能在合理的计算时间内完成求解。得益于自适应学习能力,DRL 算法不仅摆脱了固定规则的束缚,更为 KP 的变种求解提供了一种新的方法。首先,TDP-DDPG 算法依然采用 DDPG 算法的特点,通过学习一个策略网络来逼近背包问题的最优解,从而更好地从复杂且规模更大的环境中搜索解空间。同时,其经验回放机制和合理的目标网络选择能有效地解决样本相关性和训练不稳定的问题,从而使 TDP-DDPG 算法在 KP 求解中表现出较高的鲁棒性和泛化性能。尽管 DDPG 算法在 KP 求解中具有一定的优势,但其在长期依赖性以及全局交互性方面还存在一定的欠缺。为解决以上问题,进一步提高算法性能,本文将 Transformer 模块引入 DDPG 算法,从而使算法更好地捕获输入序列之间存在的依赖和交互信息。除 Transformer 模块外,加入针对背包剩余体积的动态惩罚机制(DPM)以及动态随机机制(DRM),通过对特定时刻的背包剩余体积大小进行判断,同时根据惩罚因子对特定数量的物品进行惩罚,使背包体积得到充分利用同时优化训练过程,并加入禁忌表防止重复搜索。此外,为了更好地跳出局部最优,在选择操作中加入两种优化选择方法。

最后,对不同规模维度的数据集进行实验以及算法对比,验证了 TDP-DDPG 算法具有出色的稳定性以及较高的求解精度,可以有效解决 0-1 KP。

此外，本文还解决了在不同物流服务场景中抽象出基于 KP 的变种，即异构多背包问题（HMKP）[29]。异构概念最早源于集装箱码头物流系统的抽象，旨在描述港口作业过程中，为应对其高并发性、动态性、随机性及复杂性的特征，针对码头中的堆场、泊位、场桥等关键场景进行结构设计的差异性与多样化。通过异构设计，港口在货物装卸、运输、堆码及集疏运作业中能够更灵活、高效地适应各类操作需求和不确定性因素[30]。为解决 HMKP，本文在 TDP-DDPG 算法的基础上设计基于二进制位置划分（BPD）的 BPD-DDPG 算法。实验和相关算例证明 BPD-DDPG 算法可以有效解决 HMKP。本文不仅在 0-1 KP 上提供了新的基于 DRL 的解决方案，并且尝试面向 KP 变种提出一种针对性的解决方案。

1 问题描述

1.1 0-1 KP 和 MKP

1）0-1 KP。

0-1 KP 具体描述如下：一个背包容量为 C 的背包以及 n 件物品，每件物品 $i=\{1,2,\cdots,n\}$ 具有重量 w_i 和价值 p_i 两个属性。从物品中选择一些放入背包，以使得背包中物品的总价值最大化，但是放入的物品总重量不能超过背包的容量。公式表示如式（1）、式（2）所示：

$$\max\sum_{i=1}^{n}p_ix_i \tag{1}$$

$$\text{s.t.}\sum_{i=1}^{n}w_ix_i\leqslant C \tag{2}$$

式中：x_i 表示第 i 件物品是否放入背包，$x_i=0$ 表示不放入，$x_i=1$ 表示放入。该问题的解是由 0 和 1 组成的二进制序列 $x=\{x_1,x_2,\cdots,x_n\}$，目标为找到使背包中总价值最大化的 x。

2）MKP。

MKP 是 0-1 KP 的一种扩展。在 MKP 中，每种物品不仅有重量和价值属性，还具有多个维度的属性，如体积、可放入性等。MKP 具体可描述为：一个容量为 C 的背包以及 n 件物品，每件物品 i 有 m 个属性维度，其中第 $r=\{1,2,\cdots,m\}$ 维度的属性为 a_{ir}，从物品中选择若干件放入背包中，使得背包中物品的总价值最大化，但同时满足背包的容量限制和每种属性维度的限制。除式（1）、式（2）外，增加属性维度限制公式表示如式（3）所示：

$$\sum_{i=1}^{n}a_{ir}x_i\leqslant A_r \tag{3}$$

式中：n 为物品种类的数量，表示每种物品的属性数量；A_r 表示第 r 维度的限制值。

1.2 HMKP

HMKP 作为多背包问题的一种形式，是从离散组合优化的视角出发，通过提炼多个典型物流服务场景的特征，逐渐成为复杂物流系统调度的核心问题。例如在港口作业中，船舶靠岸对泊位进行分配时，在多个结构不同泊位中各自对应可以接受停靠的船只类型和数量。将其抽象为 HMKP 时，不同结构的泊位对应多个容量不同的背包，船只对应可放入背包的物品。HMKP 与 MKP 对于装入背包的物品都有约束限制，但对于背包数量的要求不同。MKP 中只含有一个背包，而 HMKP 包含多个背包以适应物流系统中的不同服务场景。该问题的提出有利于更准确地解决物流行业中面临的控制决策问题[30]。在复杂物流服务场景中的码头泊位分配对应的 HMKP 求解架构如图 1 所示。

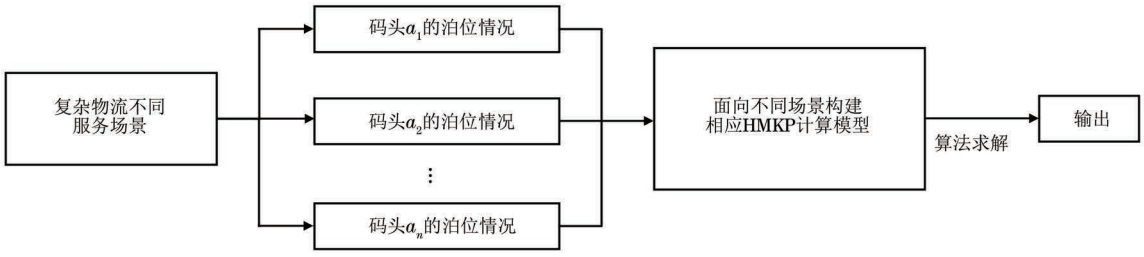

图 1 HMKP 的求解架构

Fig. 1 Solution architecture for HMKP

当复杂物流服务场景中的问题被抽象为 KP 时，对于部分或者全部的物品在选择放入哪个背包时有指定要求。针对这种情况，可以使用 0-1 二进制编码来控制其存放情况，通过给每个物品赋予是否能放入对应背包的二进制串来满足所代表的物品对分配背包的要求。

HMKP 具体可描述为：有 m 个背包和 m 堆物品，其中每堆物品用 $\{a_1,a_2,\cdots,a_m\}$ 来表示，背包容量用 V_r（$r=\{1,2,\cdots,m\}$）表示；一堆物品对应一个背包，其中物品能否放入对应的背包用 $x_{ri}=\{0,1\}$ 来表示，0 代表不能放入，1 代表可放入；w_{ri} 和 p_{ri} 代表第 r 堆物品中的第 i 件物品的重量和价

值，最终目标为使所有背包装入物品后的价值最大化。公式表示如式(4)、式(5)所示：

$$\max\sum_{r=1}^{m}\sum_{i=1}^{m}p_{ri}x_{ri} \tag{4}$$

$$\text{s. t.}\sum_{i=1}^{m}w_{ri}x_{ri}\leqslant V_r x_{ri}=\{0,1\} \tag{5}$$

2 DRL 理论

2.1 强化学习的基本原理和结构

强化学习过程中包含以下几个重要的概念，这几个概念组成了其基本结构。除智能体(agent)外，还包括状态(state)、动作(action)、策略(policy)、反馈(reward)。

强化学习一般可以描述为马尔可夫决策过程(MDP)，其决策过程如图 2 所示。

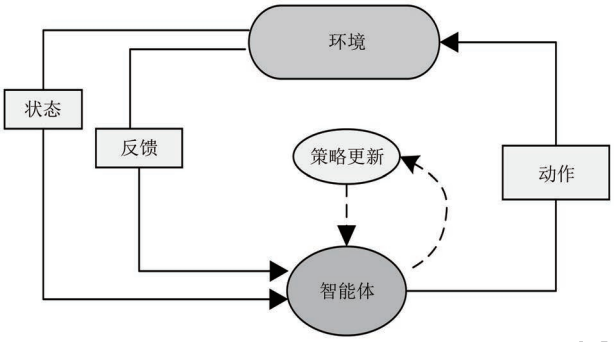

图 2 MDP 过程

Fig. 2 MDP process

在典型的 MDP 中：S 表示状态集合；S_t 表示当前状态；S_{t+1} 表示下一个状态；$s\in S$ 表示集体的状态；A 表示动作集合；$a\in A$ 表示具体动作；状态转移概率为一个条件概率分布，用 $P(s_{\text{next}}|s,a)$ 表示当前状态 S 在选择动作 a 后转移到新状态 s_{next} 的概率；$R(s,a,s_{\text{next}})$ 表示采取了 a 动作的状态 s 若转移到新状态 s_{next} 时的奖励；折扣因子 γ 用来衡量未来奖励是否重要，其中 $\gamma=\{0,1\}$。在此过程中下一个状态 S_{t+1} 仅与当前状态 S_t 有关，而与以往的状态无关，这是马尔可夫性的表现。用符号表示为 $p(s_{t+1}|s_t,s_{t-1},\cdots,s_0)=p(s_{t+1}|s_t)$。

MDP 在强化学习中的表现过程为：在时间点 t 时的状态为 S_t，之后 agent 在 S_t 状态下依据策略 π 执行动作 a_t；当执行完毕后，根据状态转移概率 $P(s_{\text{next}}|s,a)$ 来选择是否进入下一个状态 S_{t+1}。MDP 为强化学习提供了数学基础并且对模型建立有很大的帮助，也为其后续发展奠定了基础。

强化学习是通过智能体与环境进行交互并试错从而学习将环境映射到动作的一种方法。智能体通过在环境中的不断尝试来从环境中获取经验(包括奖励或惩罚)，根据奖励或惩罚信号来判别评估动作的好坏。在这个过程中，初始化的智能体并没有获取每个动作的奖惩信息，而且通过不断尝试来学习。智能体的目标为找到合适的策略，使得在执行这个策略时能从环境中获取奖励的最大化，来更好地达到目标需求。

强化学习的基本过程包括以下 6 个步骤：

1)初始化：将智能体设置为初始状态，并且具有初始的策略。

2)探索与学习：在根据经验更新的状态下，智能体会决定下一步使用已知的最佳动作还是探索新的动作来获取更好的奖励。

3)执行动作：智能体根据当前策略权衡一个动作执行。

4)执行结果：该动作会接收一个奖惩信号来反映该动作在当前状态下的好坏。

5)更新策略：智能体根据奖惩信号更新策略，使其在相同状态下选择更优动作。

6)重复上述过程：通过更新积累的经验来获取奖励最大化的策略。

将强化学习应用到背包问题时，对应的 MDP 过程包括以下 5 个部分：

1)状态空间：状态空间为每一步决策时，背包环境的所有可能状态，包括已选择的物品列表、未选择物品中符合剩余背包容量的物品列表、当前背包的剩余容量、物品的重量和价值属性。

2)动作空间：当前背包状态下的可能动作，包括根据当前策略进行选择或根据一定概率进行随机选择。

3)奖励函数：奖励函数为物品的价值密度并对其进行放缩操作。

4)状态转移：在选择一个物品后更新背包状态，包括更新已选择的物品列表、未选择物品中符合剩余背包容量的物品列表、当前背包的剩余容量、物品的重量和价值等属性。

5)终止条件：当剩余容量不能满足未被选中的任何物品或所有物品都已放入背包时终止。

2.2 DRL

DRL 相比于传统方法而言是一种更加接近人脑思维方式的一种机器学习算法，其原理融合了深度学习中拥有的感知能力和强化学习中的决策性，并且在处理复杂的决策类问题中展现出了优异的能力。DRL 作为一种端对端(end-to-end)的感控系统，其通用性表现在很多方面。学习过程可描述为：agent 在与环境交互的每个时刻中，从高

维度利用深度学习方法进行感知以获得本时刻的状态特征，之后基于回报评价价值函数并映射为相应动作，最后做出反应并继续观察。DRL 的原理框架图如图 3 所示（彩色效果见《计算机工程》官网 HTML 版，下同）。将其融入 KP 中可描述为：当 agent 选择一个动作后，背包的容量信息以及已选择的物品信息将被记录，用于为下一个动作提供参考。若信息积累到一定程度时，通过将信息进行网络训练来使 agent 选择更合适的物品装入背包。

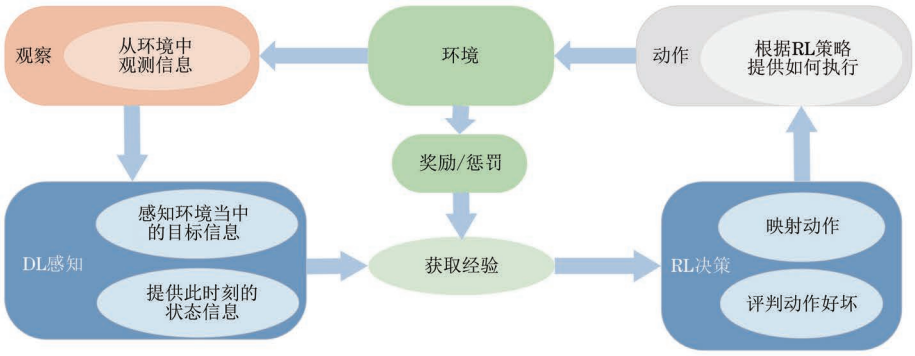

图 3 DRL 原理框架

Fig. 3 DRL rationale framework

3 求解 0-1 KP 的改进 DDPG 算法

3.1 基本 DDPG 算法

DDPG[31-32]最早是由确定性策略梯度（DDP）演化而来的。DPG 为 DDPG 提供了主要的理论基础，从理论角度出发提供了一种确定性、离线策略的策略梯度算法，并提供了其计算更新的方法。之后深度 Q 网络（DQN）[33-34]的出现解决了用神经网络拟合状态-价值函数 $Q(s,a)$ 导致的训练不稳定的问题。而 DDPG 则结合了 DPG 与 DQN 两个算法，将 DQN 中用神经网络来拟合 Q 函数这个特性应用于 DPG 的框架中，并且加入批标准化（BN）的技巧，使其实用性大大加强。

DDPG 与 DQN 的区别在于在其基础上增加了一个策略网络 $\pi(a|s)$，并且此策略为确定性的。那就表示 agent 在每一个相同状态下选择的执行动作 $a(a\in A)$ 是确定的。为了扩大算法的搜索范围、增加搜索的可能性，一般通过加入高斯噪声来处理，agent 在执行动作时通过高斯噪声可使其动作实现区域变得更为广泛。OU（Ornstein-Uhlenbeck）噪声可被视为高斯噪声的一种，相比于一般独立噪声，OU 噪声在探索过程中，可以提高在环境中探索的效率，其一般表达如式（6）所示：

$$\mathrm{d}x_t=\theta(\mu-x_t)\mathrm{d}t+\sigma\mathrm{d}W(\theta>0,\sigma>0) \tag{6}$$

式中：μ 和 σ 分别表示其均值和方差；W 可以看作是一个布朗运动。OU 噪声可进一步提高算法的探索能力。在 Actor 网络输出动作的基础上添加 OU 噪声如式（7）所示：

$$a_t=a_t+\xi_t \tag{7}$$

在整个探索过程中，噪声的添加服从由强到弱的变化趋势。这种变化趋势有利于在前期增加探索概率，获取范围更广的经验积累，在后期转而减少探索概率，依据前期积累的经验强化算法本身的利用能力，从而实现算法训练由前到后的平稳过渡，提高算法的收敛效率。

DDPG 算法依据演员-评论家（Actor-Critic）网络框架进行运作。Actor 与 Critic 网络分工合作，前者调控 agent 的行为策略，后者依据评估标准评价动作的好坏。Actor 网络会根据评估的情况来改变策略以获取更好的回馈，高价值动作的积累会形成良性的循环，使算法整体向效果更好的方向发展。

这个过程算法将不断产生状态、动作、奖励以及下一状态的经验信息，并且存入经验池 (s_t,a_t,r_t,s_{t+1}) 中。当经验累积到一定量程度时，算法将抽取一定量较小的样本，按照梯度下降的方式训练网络。

DDPG 算法中采用了 4 个网络，分别为初始的 Actor 和 Critic 网络以及与初始 Actor、Critic 网络参数相同的两个目标网络，如图 4 所示。

初始的 Actor 网络在与环境的交互中获取状态信息 s，同时在环境中加以动作表达 a。当同时将这两个信息传入初始 Critic 网络时输出当前状态和动作的 Q 值。Q 网络是 Critic 网络的参数化表示，用于描述在长期一段时间的条件下本行为的回报获取。Critic 网络的更新是从经验池中取较小样本进行训练得到 Q 值，然后将 Actor 网络训练得出的下一动作 a_{t+1} 一起传入目标 Critic 网络从而得到目标 Q 值，公式描述如式（8）所示：

$$Q_t=r+\gamma Q(s_{t+1},a_{t+1}) \tag{8}$$

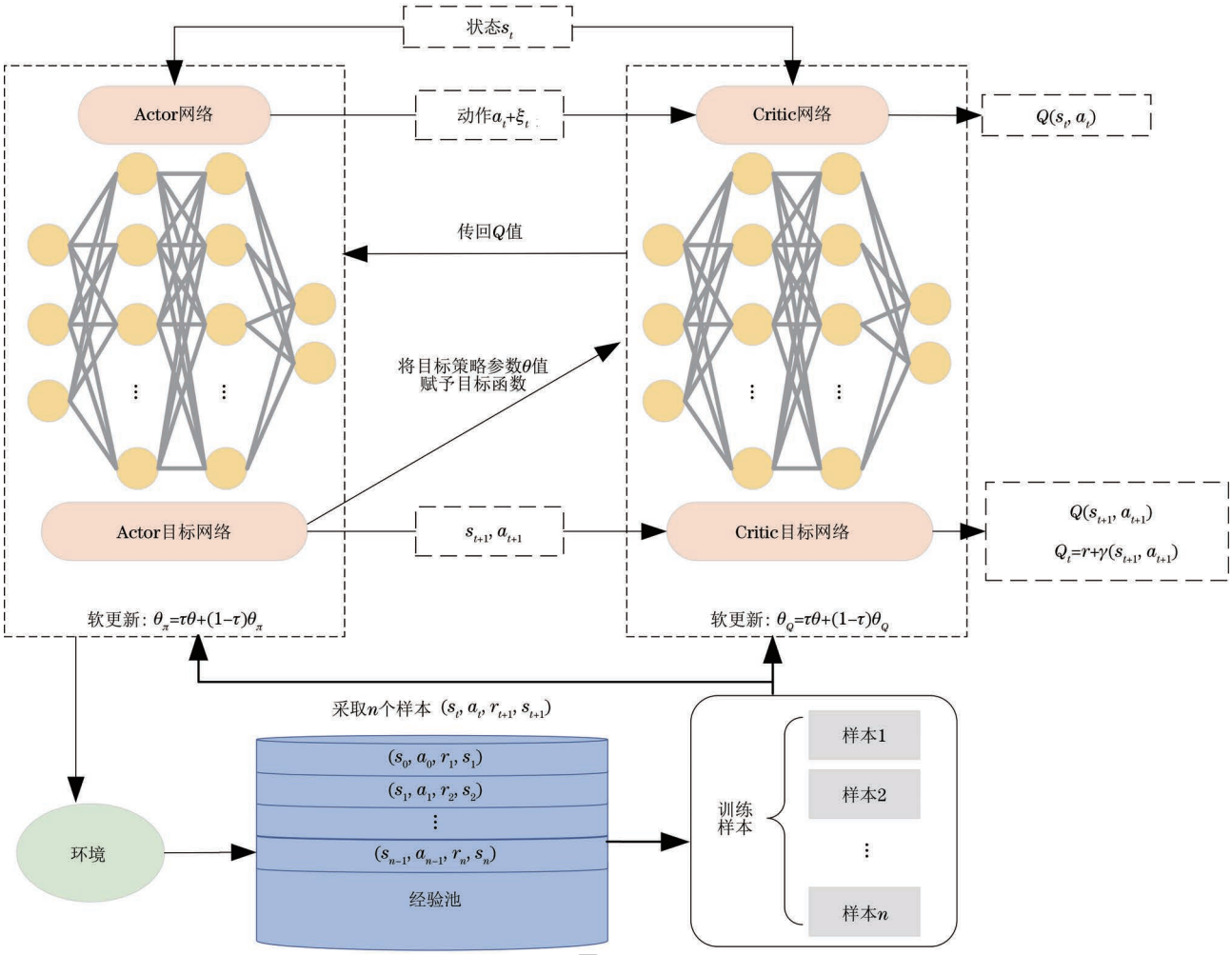

图 4 Actor-Critic 结构图

Fig. 4 Actor-Critic structure diagram

式中:γ 表示折扣因子,其作用为获取当前回报时同时将未来可能的回报也考虑进去。

Actor 网络参数的更新涉及针对策略梯度 θ_π 的计算,采用蒙特卡罗方法对 Actor 网络参数更新。目标函数 J 对 θ_π 梯度计算公式如式(9)所示:

$$\nabla_{\theta^\pi} J(\theta_\pi)\approx\frac{1}{N}\sum_{i=1}^{N}(\nabla_a Q(s,a,\theta_Q)\nabla_{\theta^\pi}\mu(s_t\theta_\pi)) \tag{9}$$

Critic 网络参数的更新涉及针对价值梯度 θ_Q 的计算,目标函数 L 对 θ_Q 梯度计算公式如式(10)、式(11)所示:

$$y=r+\gamma Q(s,\mu(s_{t+1},\theta_\pi),\theta_{Q(t+1)}) \tag{10}$$

$$\nabla_{\theta_Q} L(\theta_Q)\approx \frac{1}{N}\sum_{i=1}^{N}(y-Q(s,a,\theta_Q)\nabla_{\theta_Q} Q(s,a,\theta_Q)) \tag{11}$$

DDPG 算法中 Actor 和 Critic 网络均采用软更新的方式,利用降低网络参数的更新速度来使目标网络的产出值更加平稳,提高网络的稳定性。目标网络的参数更新公式如式(12)、式(13)所示:

$$\theta_{\pi(t+1)}=\tau\theta_\pi+(1-\tau)\theta_{\pi(t+1)} \tag{12}$$

$$\theta_{Q(t+1)}=\tau\theta_Q+(1-\tau)\theta_{Q(t+1)} \tag{13}$$

式中:τ 为学习率,表示为软更新参数,其取值范围为大于 0 且远小于 1,一般为 0.001、0.000 1 等,本文采用的 τ 值为 0.001。

3.2 DDPG 算法改进

原始的 DDPG 在处理高维度、复杂的动作空间时具有很好的优化性能,并且利用深度神经网络中的近似函数和策略函数在大规模的状态处理空间中有较强的表达性。但是原始的 DDPG 算法在解决 KP 时往往收敛不到最优的解,或者说通常只能收敛到一般的解,且容易陷入局部最优。这个问题是由于 DDPG 算法在训练时的高复杂性以及高度敏感性造成的。原始 DDPG 算法对大量超参数的依赖性很强,对于数据的变化高度敏感,包括网络的学习率、更新率、结构等。不合适的超参数选择对算法具有牵一发而动全身的后果。

根据经过大量实验得出的超参数集合进行网络训练时可以达到较为理想的训练效果,但是针对

KP 来说最好只能达到次优解。

通过对奖励进行缩放，可以提高网络进行选择操作时的准确性，从而达到提高算法精度的目的。当算法网络训练成熟时，根据自身所学习到的经验并结合回报可以从数据集中选取回报积累高的物品装入背包，如式(14)所示：

$$R=(v_{pr}/V_{cap}-k)\cdot w \tag{14}$$

式中：R 代表回报；v_{pr} 代表物品价值；V_{cap} 代表物品体积；k 决定将一些低价值的物品放入低优先级队列，一般设置为背包最大价值与背包最大容量的比值；w 代表扩大回报的权重，将剩下的高价值密度物品价值放大，使物品价值差距更加清晰，便于选择。为防止低价值密度物品长期不被选择，保持算法的探索性能，$R\in(o,t)$，其中 t 与 w 的取值具有关联性，在众多物品中，价值密度最大的物品其数值通常稳定在 10 附近，所以将 t 和 w 设置为 10 既可以将所有物品的价值密度缩放到一个合理的范围内，又可以防止回报过高导致梯度爆炸而影响算法稳定性。

为进一步提高算法的精度，通过在算法中融入改进 Transformer 模块，并加入 DRM 和 DPM，以及选择最优方法来解决。

3.2.1 动态随机机制

DRM 的操作过程为：当算法在选择物品放入背包时，依据随机 s_R 来选择是否根据已学习的经验来选择物品放入背包，具体过程如图 5 所示。

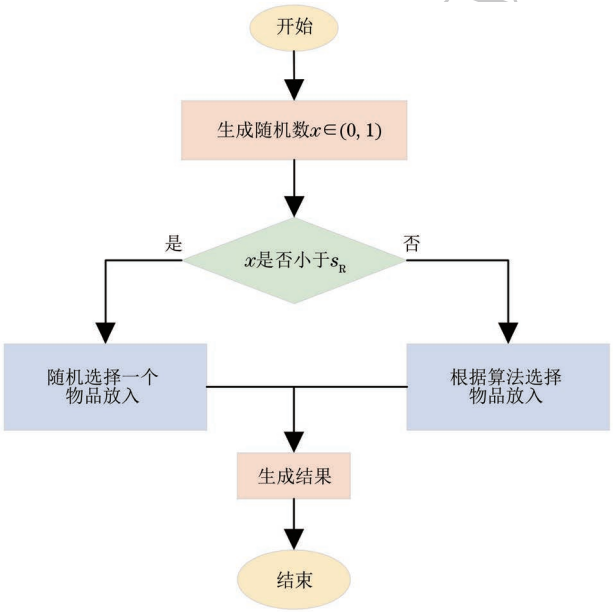

图 5 DRM 的操作过程

Fig.5 DRM operation process

当设置 $s_R=0.01$ 时表示在执行选择动作时，有 1%的概率不根据算法本身做出选择，而是从剩余物品中随机选择一个放入背包。DRM 是随着训练集训练一轮的次数(epoch)的动态变化过程，设置 s_R 为递减过程且数值根据训练效果在 0～0.01 之间变换，前期设置大的探索率是为了更丰富地积累经验池，后期逐渐减小到 0 是为了防止随机选取物品影响收敛精度。

3.2.2 动态惩罚机制

按照无惩罚机制进行选择时会出现贪婪地选择价值密度最大的物品，在算法训练中后期的选择过程中，由于背包容量越来越小，虽然有满足剩余容量的物品，但算法依旧依据高奖励选取那些不满足剩余容量的大体积物品，导致算法提前跳出结果，陷入局部最优。

DPM 的设置需要一个阈值 p_x，该值控制算法在多少次迭代内实施惩罚，本文设置 $p_x=150$。其运作过程可描述为：首先计算目前未被选择的物品价值密度，然后给予不满足背包剩余容量对应的物品极大惩罚，保证不再选择不符合剩余容量的物品。这样可以排除训练后期算法在每一步进行选择时的不可行解，防止算法过早地跳出结果，陷入局部最优的情况。

同时设置惩罚因子 p_e 来依据物品的体积大小惩罚剩下满足容量的物品，如式(15)、式(16)所示：

$$\lambda_{penalty\_factor}=P_e\times V_z \tag{15}$$

$$R=R-\lambda_{penalty\_factor} \tag{16}$$

式中：V_z 代表剩余体积；R 代表回报惩罚因子的大小设置为随着 epoch 进行而递减的动态变化，本文设置为由 0.05 到 0。上述设置的作用是：一方面当背包剩余容量越来越小时，在价值密度相似的情况下可以有一定概率选择体积较小的物品，有利于充分利用背包空间；另一方面也防止了算法选择时过度选择价值密度最大的物品，由于过于贪婪陷入局部最优。

为进一步优化选择，在算法选择过程中加入两个优化方案。首先，当算法选择一个物品装入背包前，先搜索剩余未被选入背包中的与其同体积容量的物品，通过价值比较选择价值最大的物品。当物品增多时，相同体积容量的不同物品会增多，装入这些物品中的任意一个对背包剩余体积容量影响是一样的，选择价值最大的那个可以直接提升背包当前总价值，并且对后续选择没有任何影响，其作用可以直接提升算法的精度。此方法可以提高神经网络在选择时的准确率，帮助算法跳出局部最优。

此外，为了更进一步防止算法陷入局部最优，提

高算法精度，将算法的选择范围由指定的物品选择优化为在被选物品的体积范围内进行选择。过程可描述为：当神经网络选择一个物品后暂且先不放入背包；将与此物品体积容量相差不超过 h 的所有物品进行搜索，本文设置 $h=5$，一般根据算例类型进行适当调整；最后将搜索到的物品按照价值密度进行排序，选择价值密度最大的物品作为最终选择对象。与单纯地选择指定物品相比，此方法可以更灵活地在与其近似重量的物品中选取价值密度更大的物品，其目的也是跳出局部最优，提高算法收敛精度，具体 DPM 过程如算法 1 所示。

算法 1 DPM

输入 阈值 p_x，剩余体积 V_z，物品体积 V_i

输出 最优物品 V_o

1. For epoch←x to epochs do：
2. if p_x：
3. 计算所有未被选择的物品体积 V_i
4. if $V_z<V_i$：
5. 对不满足剩余体积的物品 V_i 进行极大惩罚
6. return 0
7. else
8. 对满足条件的物品依据惩罚因子 p_f 进行惩罚
9. 从体积容量相差 h 范围内选择价值密度最大物品 V_m
10. 搜索与物品 V_m 体积相同的物品序列
11. 从序列中计算出价值密度最大的物品 V_o 作为输出
12. return V_o
13. break
14. End For

3.2.3 改进 Transformer 模块

Transformer 网络模型是一种深度学习架构，最初在 2017 年被引入。在后续发展中，Transformer 网络模型在计算机视觉领域得到了广泛应用，并展现出强大的性能。在处理动作空间方面，Transformer 网络模型通常需要结合强化学习算法来进行训练。本文将 Transformer 模块嵌入到 DDPG 算法中用于处理状态表示和动作策略的生成，而 DDPG 算法本身则用于近似值函数来评估以及优化策略。通过 Transformer 网络模型中的多头自注意力(MSA)机制，使算法模型能准确地获取背包中物品之间的关联性和动态变化，从而更好地实现 KP 的优化目标。

Transformer 网络模型中的 MSA 机制能对 DDPG 算法中的 Actor 网络和 Critic 网络产生影响，用来处理状态信息。在 Actor 网络中，MSA 机制用来处理背包中物品状态的信息时，更加注重全局特征之间的联系。通过 MSA 机制，网络可以在生成动作之前对输入状态进行加权聚合，使状态之间的关联性放大，从而生成更准确的动作策略。在 Critic 网络中，MSA 机制摒弃传统单头注意力机制中网络只能聚焦于状态信息的一个方面这个缺点，通过并行地计算多个注意力头，使网络对每个注意力头所关注到的状态信息的不同部分进行理解，其计算结构如图 6 所示。MSA 具体参数设置为：注意力头数为 8，输入维度为物品个数，查询向量(query，$\boldsymbol{Q}$)、键向量(key，$\boldsymbol{K}$)、值向量(value，$\boldsymbol{V}$)空间维度等于输入维度除以注意力头数，编码完成后送入全连接层(层数为 4)。

MSA 机制中定义一系列线性变换操作和注意力权重的缩放因子，如输入维度、头数、头维度等。通过线性变换将输入数据映射到查询向量、键向量、值向量空间。以下对 3 个向量进行描述：

1)query：通过计算注意力权重来表示元素与元素之间的相关性，用于与其他 key 的比较来确定受关注程度。

2)key：用于衡量查询向量与其他元素的相似程度，从而确定不同元素之间的关联性。

3)value：用于根据计算得到的注意力权重对输入数据进行加权求和，以产生最终输出。

将每个头的 $\boldsymbol{Q}$、$\boldsymbol{K}$ 和 $\boldsymbol{V}$ 通过“view”操作重塑张量的形状，以便进行矩阵乘法操作。接下来，计算 $\boldsymbol{Q}$ 和 $\boldsymbol{K}$ 之间的点积，并将结果除以缩放因子以缩放到合适的范围。之后使用 Softmax 函数计算得到注意力权重，并且用于加权求和 $\boldsymbol{V}$。同时，将注意力权重应用到 $\boldsymbol{V}$ 上，并对每个头的输出进行重新排列和重塑，以便后续处理。最后，将得到的注意力输出通过线性变换操作得到最终的输出结果。具体过程如式(17)～式(20)所示：

$$S_c=\frac{\boldsymbol{Q}\cdot\boldsymbol{K}}{\sqrt{d_k}} \tag{17}$$

$$a_w=\mathrm{Softmax}(S_c) \tag{18}$$

$$o_{sa}=a_w\cdot\boldsymbol{V} \tag{19}$$

$$O_u=o_{sa}\cdot\boldsymbol{W}_o \tag{20}$$

式中：S_c 为注意力得分；a_w 为计算注意力权重；o_{sa} 为计算自注意力输出；O_u 为最终输出；d_k 是头维度；$\boldsymbol{W}_o$ 是输出矩阵的权重矩阵。

在原始算法中嵌入改进 Transformer 模块后，算法获取背包中物品之间的关联性能力有所加强。模型在选择动作策略时，不再只关注单个物品的价值属性，同时还考虑到物品与物品之间组合后产生的相互作用的关联关系。具体表现在以下 3 个方面：

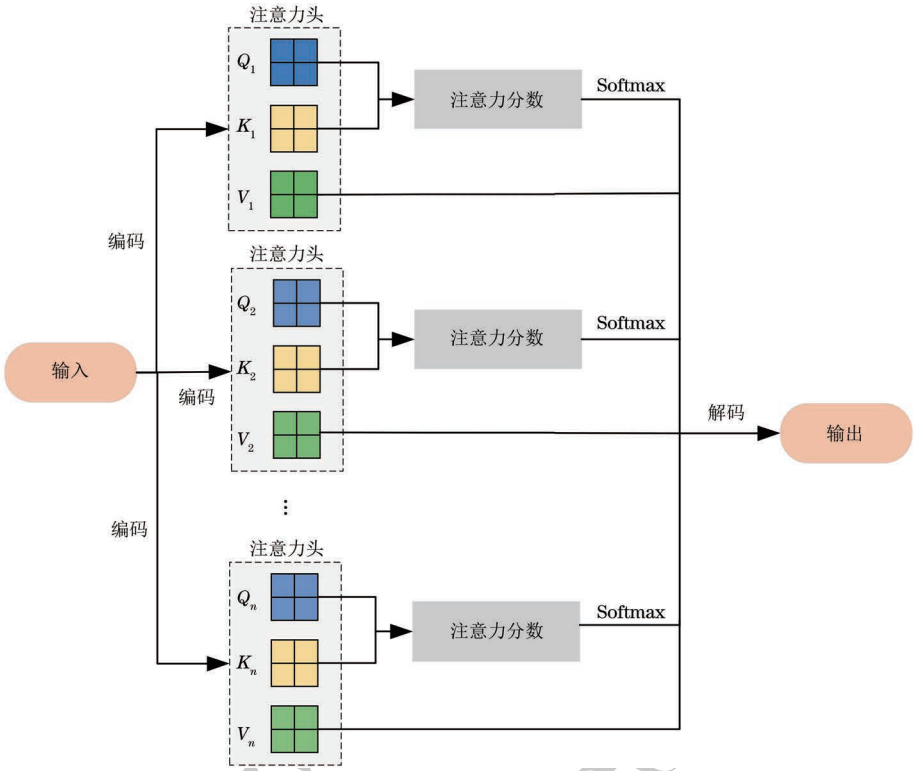

图 6 MSA 计算结构

Fig. 6 MSA computational architecture

1)全局关联性:MSA 机制将学习到的每个物品信息经过注意力权重进行加权组合,从而在全局角度下观察不同物品之间的关联性,从中获取决策。相比于经典算法中简单地由物品属性做出决策的方式,能更加准确地捕捉到物品之间的关系。

2)动态关联性:改进算法能够根据当前的背包状态调整注意力权重,从而更好地识别不同背包状态下物品之间的关联。相比于经典算法中基于固定的规则来做出决策更加灵活,可以更好地适应不同情形,从而做出动态变化。

3)非线性关联性:改进 Transformer 模块中的 MSA 机制和非线性激活函数能够处理物品之间的非线性关系,使得物品之间的复杂关系清晰化,进而提高模型识别的准确性,提高决策效率。

由于改进机制使物品之间的关联性能力增强,使算法在训练和推理方面的效率也得到提高。具体体现在模型的并行处理能力、高效简洁的计算结构、参数更新效率以及内存的高效利用。

模型在训练时,MSA 机制的引入增强了模型对状态空间的理解能力,使其更好地获取其状态之间的依赖关系。MSA 机制可以通过对状态的自我关联,提取出更丰富的特征表示。这有助于模型更准确地评估出不同动作的价值,加速训练进程。此外,其还可以对不同的多个物品信息进行并行处理,并行处理能力使模型的计算结构简洁且高效,使计算资源和内存空间得到充分利用,避免了资源浪费和内存溢出的问题。

MSA 机制同样提高模型的推理效率,其获取物品长期依赖关系的能力,能使模型在推理时考虑长期的历史状态信息。此外,模型通过反向传播更新参数,利用 MSA 机制生成的梯度信息高效地更新模型参数,加快收敛进程。

综上所述,DRM、DPM 的引入使模型在每个阶段的训练和推理更加清晰,其不仅有利于算法跳出局部最优,而且增加了算法的收敛效率。MSA 机制的引入使模型在训练时更加明确不同物品之间组合关系的优劣,从而在长期历史状态信息中更准确地获取执行动作,通过对算法的改进使算法在训练和推理方面更加高效。

3.2.4 禁忌表

解决 KP 时,重点要关注在给定背包容量的情况下选择哪些物品,那么就可能出现在背包中放入完全相同的两组物品这种情况,影响算法的全局搜索。加入禁忌表来记录搜索过程可以有效解决重复搜索问题,使算法更好地跳出局部最优。

禁忌表主要的目的是防止算法在搜索的过程中出现循环从而使算法陷入局部最优。在 KP 中,通过赋予禁忌表一定容量,用来记录算法在搜索过程

中放入的物品组合，同时禁止表内的物品在近期内重复出现。同时，当迭代到一定轮次时，禁忌表会将记录内容进行释放，使之前被存入禁忌表中的物品组合可以重新被搜索到。禁忌表策略既保证了算法良好的全局搜索和跳出局部最优的能力，又使算法具有一定的灵活性和自适应性。

4 求解 HMKP 的 BPD-DDPG 算法分析

HMKP 作为多背包问题的一种特殊形式，在其算法求解方式上与传统的解题方法增加了约束条件，并且要求算法可以在复杂度较高情况下进行处理。因此，在算法的结构设计上增加了较大的难度。

TDP-DDPG 算法在处理 KP 时是根据固定的背包来选择装入背包的固定物品的，因此可以将背包中的物品增添一个二进制位来控制其是否可以装入某些背包，此思想也更贴合现实物流领域的运作流程。以航运物流为例，将船舶和港口抽象为物品和背包，通过融入添加二进制位来控制物品存放位置的思想形成 BPD-DDPG 算法来处理 HMKP。

4.1 BPD-DDPG 算法结构

算法在结构设计时根据背包中的不同物品针对可选择放入的背包进行二进制划分，以两个背包为例，每一堆物品中的每个物品都有对背包 1 和背包 2 的二进制编码，如图 7 所示。根据所需可以将两堆物品进行任意组合，不再局限于指定物品放入指定背包这个条件。

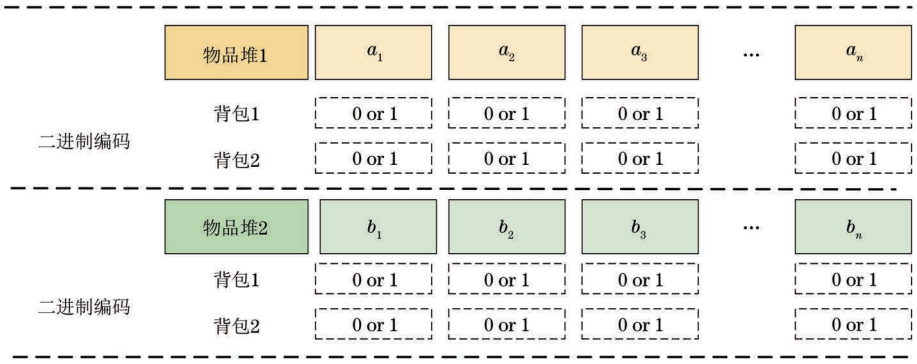

图 7 二进制划分

Fig. 7 Binary division

BPD-DDPG 算法在对每一个背包进行处理时都相当于一个简单的 KP，HMKP 的数学模型如式(21)～式(23)所示：

$$g_{ij}=O_i-V_{ij} \tag{21}$$

$$G_i=\sum_{j=1}^{m} g_{ij} \tag{22}$$

$$G_{\min}=\min\{G_1,G_2,\cdots,G_n\} \tag{23}$$

式中：$O_i(i=\{1,2,\cdots,n\})$代表每个背包的最优值；$V_{ij}(j=\{1,2,\cdots,m\})$代表当前算法得出的背包的物品价值总和；$g_{ij}$ 代表每次算法得出的最优值与 V_{ij} 之间的差值；G 则代表所有背包的差值 g 之和。

式(21)表示求得的多背包中每个背包当前得出背包物品价值总和与每个背包最优值之间的差值；式(22)表示将求得的差值求和；式(23)的目标是求得最小的 G 值。

4.2 BPD-DDPG 算法框架

BPD-DDPG 是在 TDP-DDPG 的结构基础上为背包内的每个物品根据位置进行二进制编码，以此来实现 HMKP 中物品的存放关系，其过程如算法 2 所示，其中 epoch 代表算法的运行代数，并且作为算法的唯一终止条件。

算法 2 BPD-DDPG

输入 运行次数 runs，数据路径 pathname

输出 Gap_{min}

1. For run←1 to runs do：
2. 初始化 price，heads，batch
3. For epoch ← 1 to epochs do：
4. 重置环境
5. 获取初始状态
6. 确定一个物品的二进制位并进行选择
7. 基于 R_t 选择下一执行动作 a_{t+1}
8. 更新下一个状态序列 S_{t+1} 来获取奖励 R_t
9. 存储<S_t，a_t，S_{t+1}，R_t>信息到经验池 R_0
10. 更新当前状态
11. End For
12. End For

在算法 2 中，runs 代表运行的总次数，pathname 代表数据集的路径，price 表示每一代的最优价值，heads 和 batch 代表多头自注意力头数及训练步长，R_0 代表经验池，S 和 a 为状态和动作。

图 8 是以解决 3 个背包组合为例来求解 HMKP 的 BPD-DDPG 算法框架图。通过并行求解 TDP-DDPG 算法来得出每个背包的 Gap 值，同时通

过存储评估得出每次训练得到的最小 Gap 总值 G_n 而达到实验目的。BPD-DDPG 以求解所求背包的整体最小 Gap 值为目的，从而达到整体最优。

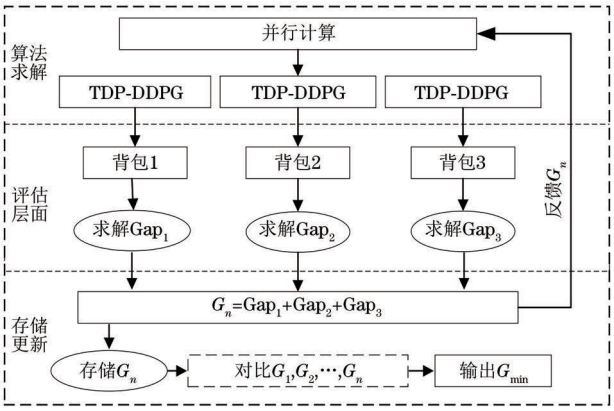

图 8 BPD-DDPG 算法框架

Fig. 8 BPD-DDPG algorithmic framework

4.3 算法的时间复杂度分析

算法的时间复杂度作为衡量计算成本的核心指标之一，可以科学地评价算法的效率与可行性。未引入 DPM 时的算法单次迭代的最大时间复杂度为 $O(n)$，而引入 DPM 的算法的时间复杂度为 $O(n)$。TDP-DDPG 算法与 BPD-DDPG 算法在时间复杂度上都为 $O(n+n)$，可表示为 $O(n)$。

引入求解 KP 的元启发式算法：基于转移函数的二元爬行搜索算法(BinRSA)[35]、混合贪婪遗传算法(HGGA)[36]以及二进制帝国竞争算法(BICA)[29]与 BPD-DDPG 算法对比单次迭代最大时间复杂度，如表 1 所示。可以看出，BPD-DDPG 算法的时间复杂度与 BICA、HGGA 算法相同，比 BinRSA 算法低。值得注意的是，TDP-DDPG 算法与 BPD-DDPG 算法的最大时间复杂度相同，说明 BPD-DDPG 的结构没有增大算法的计算成本，所以其结构的设计在理论上具有合理性。

表 1 不同算法的时间复杂度对比

Table 1 Comparison of time complexity of different algorithms

算法	时间复杂度
BPD-DDPG	$O(n)$
BICA	$O(n)$
BinRSA	$O(n^2)$
HGGA	$O(n)$

5 实验与分析

为验证上述所提到的 TDP-DDPG 算法以及未加入改进 Transformer 模块前的 DP-DDPG 算法的性能，分析基于算法上的各个超参数的选择合理性和适配性，分别用物品数量不同、问题规模不同、数据相关性不同的 3 个经典 0-1 KP 测试集进行实验。

测试集 1 中包含 10 个算例(kp1～kp10)[29]，物品的数量维度涵盖了低维度、中维度、高维度 3 种类型，由 kp1 到 kp10 由低到高排序。测试集 2 中包含 9 个算例(kp11～kp19)[37]，包含 10～100 个物品维度，且在 100 个物品维度的算例中部分问题规模较大。

在前两个测试集的基础上进一步在更高物品数量维度的测试集 3 上进行探讨。测试集 3 中包含 20 个算例(kp20～kp39)[29]，涵盖了无相关、弱相关、强相关、完全相关等数据类型。最后在大规模测试集 4 上进行实验测试。物品数量维度越高、问题规模越大，算法运行复杂度越高。最后验证 BPD-DDPG 算法能够有效求解 HMKP 并且具有较好的实验结果。

本文实验的超参数设置如下：Actor 网络和 Critic 网络学习率为 0.001；经验回放缓存的最大长度为 10 000；折扣因子为 0.95；在选择动作时的随机探索概率为 0.01；训练时的批量大小为 64；经验回放缓存中存储的样本数量达到 128 时开始训练。

以上实验均在 PyCharm 2020.1.3(Professional Edition)环境中编写与测试，模型代码以 PyTorch 框架为基础进行搭建运行。处理环境为 13th Gen Intel® Core™ i9-13900HX 2.20 GHz、GPU(NVIDIA GeForce RTX 4060)、64 位 Windows 11 操作系统，编程语言采用 Python 3.9。

5.1 TDP-DDPG 算法在测试集 1 上的性能对比

实验在测试集 1 上将每个算例在训练时分为 10 组，每组分别训练迭代 200 次，并且单独保存训练结果。选取以牵制关系为优化驱动力的新型仿生算法：牵制平衡算法(CBA)[38]作为主要对比算法，加以标准粒子群算法(PSO)[39]，在 PSO 基础上采用概率映射方式且具有很强全局搜索能力的离散二进制粒子群算法(BPSO)[39]，以及模拟退火算法(SA)[39]。

表 2 所示为各个算法的对比结果，表中最优值以加粗的形式标出，并且在后续测试集依然采用这种标注方式，表中符号—代表文献中没有给出最优值。本文算法测试结果对比如表 3 所示，其中成功率为求得最优值的概率。从表 2 中可以看出，在测试集 1 中的 10 个算例中，原始 DDPG 算法只有 2 个算例可以达到最优，而加入 DRM 以及 DPM 机制后的 DP-DDPG 以及在此基础上将改进 Transformer 模块嵌入后的 TDP-DDPG 算法在求解时全部可以达到最优解。由于求解难度是随着物

品维度的上升而增高的，原始 DDPG 算法仅能在低维度上达到最优，而 DP-DDPG 算法和 TDP-DDPG 算法不仅在中低维度上可以达到最优，在高维度上依旧可以稳定寻找到最优解。由此可见，改进后的两个算法在收敛精度方面得到了很大提升。由表 3 可知，TDP-DDPG 算法相较于 DP-DDPG 算法在最优解、最差解、均值解和标准差方面都相同，在寻优精度上表现优异。

表 2 TDP/DP-DDPG、DDPG 算法在测试集 1 上的实验数据对比

Table 2 Comparison of experimental data of TDP/DP-DDPG, DDPG algorithms on test set 1

算例	问题规模	理想最优解	PSO	BPSO	SA	CBA	DDPG	DP-DDPG	TDP-DDPG
kp1	10	295	**295**	**295**	**295**	**295**	**295**	**295**	**295**
kp2	15	481.07	475.48	**481.07**	**481.07**	**481.07**	**481.07**	**481.07**	**481.07**
kp3	20	1 024	978	**1 024**	**1 024**	**1 024**	**1 024**	**1 024**	**1 024**
kp4	23	9 767	9 756	**9 767**	**9 767**	**9 767**	**9 767**	**9 767**	**9 767**
kp5	50	3 103	2 897	3 061	3 038	**3 103**	3 080	3 103	**3 103**
kp6	50	4 882	—	—	—	**4 882**	4 844	**4 882**	**4 882**
kp7	50	16 102	13 044	15 332	15 986	**16 102**	15 977	**16 102**	**16 102**
kp8	60	8 362	6 594	7 973	**8 362**	8 354	**8 362**	**8 362**	**8 362**
kp9	80	5 183	4 528	4 538	5 074	**5 183**	5 178	**5 183**	**5 183**
kp10	100	2 660	—	—	—	2 645	2 631	**2 660**	**2 660**

表 3 TDP/DP-DDPG 算法在测试集 1 上的测试结果

Table 3 Results of TDP/DP-DDPG algorithm on test set 1

算例	问题规模	理想最优解	算法	最优解	最差解	均值解	标准差	成功率/%
kp1	10	295	TDP-DDPG	295	295	295	0	100
			DP-DDPG	295	295	295	0	100
kp2	15	481.07	TDP-DDPG	481.07	481.07	481.07	0	100
			DP-DDPG	481.07	481.07	481.07	0	100
kp3	20	1 024	TDP-DDPG	1 024	1 024	1 024	0	100
			DP-DDPG	1 024	1 024	1 024	0	100
kp4	23	9 767	TDP-DDPG	9 767	9 767	9 767	0	100
			DP-DDPG	9 767	9 767	9 767	0	100
kp5	50	3 103	TDP-DDPG	3 103	3 103	3 103	0	100
			DP-DDPG	3 103	3 103	3 103	0	100
kp6	50	4 882	TDP-DDPG	4 882	4 882	4 882	0	100
			DP-DDPG	4 882	4 882	4 882	0	100
kp7	50	16 102	TDP-DDPG	16 102	16 102	16 102	0	100
			DP-DDPG	16 102	16 102	16 102	0	100
kp8	60	8 362	TDP-DDPG	8 362	8 362	8 362	0	100
			DP-DDPG	8 362	8 362	8 362	0	100
kp9	80	5 183	TDP-DDPG	5 183	5 183	5 183	0	100
			DP-DDPG	5 183	5 183	5 183	0	100
kp10	100	2 660	TDP-DDPG	2 660	2 660	2 660	0	100
			DP-DDPG	2 660	2 660	2 660	0	100

为了解 7 个算法之间对比的差异性，在置信水平为 0.05 的条件下，使用非参数 Wilcoxon 秩和检验，结果如表 4 所示。可以看出，TDP-DDPG 算法和 DP-DDPG 算法对 DDPG、PSO、BPSO、SA 算法在收敛精度上具有显著优势，对于 CBA 算法虽然在低维度和中维度相互持平，但在高维度上具有显著优势。

表 4 TDP-DDPG 与 6 个对比算法在测试集 1 上的秩和检验测试结果

Table 4 Test results of rank sum test between TDP-DDPG and 6 comparison algorithms on test set 1

算法对比	Better	Equal	Worse	p 值
TDP-DDPG vs DDPG	5	5	0	0.043 1
TDP-DDPG vs DP-DDPG	0	10	0	1.000 0
TDP-DDPG vs CBA	2	8	0	0.179 7
TDP-DDPG vs SA	3	5	0	0.108 8
TDP-DDPG vs BPSO	4	4	0	0.067 8
TDP-DDPG vs PSO	7	1	0	0.017 9

为了进一步验证算法的性能对比，以及验证算法在测试集 1 上得出的最优解是否正确，选取其他算法对测试集 1 得出的最优解进行验证，如表 5 所示。表中列出了基于互相学习的队列智能算法(CI)[40]，包含位置更新和遗传变异的全局协调搜索算法(NGHS)[38]，基于自然发展过程的和声搜索算法(HS)[38]，以及在此基础上加入动态更新过程的改进和声搜索算法(IHS)[38]，基于贪婪策略的风驱动优化算法(CWDO)[41]，基于回声定位的蝙蝠算法(BA)[42]，新颖复值编码的蝙蝠算法(CPBA)[42]，利用贪婪算子改造的贪婪遗传算法(GGA)[42]，贪婪粒子群优化算法(GPSO)[42]。还有一系列启发式算法，如二进制狮群算法(BLSO)[37]、混合蝙蝠算法(HBA)[43]、标准蚁群算法(ACO)[43]、引入侦查子群的蚁群算法(ACOIS)[43]、自适应元胞粒子群算法(ACPSO)[44]、二进制蝙蝠算法(BBA)[45]等算法对于测试集 1 中各算例最优解的值对比。可以看出，有部分算法的训练结果在测试集 1 上的最优值比 TDP-DDPG 算法差，剩下的测试结果在最优值上和 TDP-DDPG 保持持平，同时没有测试结果在最优值上比 TDP-DDPG 算法高。那就证明表 1 中所列出测试集 1 的理想最优解是正确无误的，证明所训练得出的数值就是测试集 1 中每个算例的现实最优解。

表 5 TDP/DP-DDPG 与其他算法在测试集 1 上的实验最优值对比

Table 5 Comparison of experimental optimal values of TDP/DP-DDPG with other algorithms on test set 1

算例	问题规模	理想最优解	DP-DDPG	TDP-DDPG	其他算法得出的最优解
kp1	10	295	**295**	**295**	CI(**295**)，HS(**295**)，CWDO(**295**)，IHS(**295**)，NGHS(**295**)
kp2	15	481.07	**481.07**	**481.07**	CI(**481.07**)，HS(**481.07**)，CWDO(**481.07**)，IHS(**481.07**)，NGHS(**481.07**)
kp3	20	1 024	**1 024**	**1 024**	CI(**1 024**)，HS(**1 024**)，CWDO(**1 024**)，IHS(**1 024**)，NGHS(**1 024**)
kp4	23	9 767	**9 767**	**9 767**	CI(9 759)，HS(**9 767**)，CWDO(**9 767**)，IHS(**9 767**)，NGHS(**9 767**)
kp5	50	3 103	**3 103**	**3 103**	BA(3 014)，GGA(3 028)，GPSO(**3 103**)，CPBA(**3 103**)
kp6	50	4 882	**4 882**	**4 882**	BLSO(**4 882**)，BBA(**4 882**)，ACPSO(**4 882**)，HBA(**4 882**)
kp7	50	16 102	16 051	**16 102**	BA(**16 102**)，GGA(15 469)，GPSO(**16 102**)，CPBA(**16 102**)
kp8	60	8 362	**8 362**	**8 362**	ACO(7 775)，ACOIS(**8 362**)
kp9	80	5 183	**5 183**	**5 183**	BA(5 176)，GGA(4 751)，GPSO(5 176)，CPBA(**5 183**)
kp10	100	2 660	2 645	**2 660**	BLSO(**2 660**)，BBA(**2 660**)，ACPSO(**2 660**)，HBA(**2 660**)

以下对原始 DDPG 算法的改进策略进行验证说明，包括原始 DDPG 算法，以及加入 DRM 和 DPM 的 DP-DDPG 算法，还有在 DP-DDPG 基础上融入 Transformer 模块的 TDP-DDPG 算法。

图 9 显示了 3 个算法在 kp8 上的收敛曲线图，其中横坐标为迭代次数，纵坐标为训练所得值与最优值之间的误差值。图中，原始 DDPG 算法收敛最慢，训练过程幅度较大，无法较好地跳出局部最优的情况。TDP-DDPG 算法和 DP-DDPG 算法的收敛曲线相较于原始 DDPG 算法可以明显快速收敛到 0，且可以稳定在其周围波动，波动的原因是探索学习的原因，属正常现象。这说明 DRM 和 DPM 的加入能够使算法快速收敛到最优，且稳定保持。此外，TDP-DDPG 算法在迭代次数上优于 DP-DDPG 算法，说明融入 Transformer 模块可以使算法更早地收敛，且保持了高收敛精度，但由于数据处理变得更为复杂，在迭代一次的时间上 TDP-DDPG 算法略高于 DP-DDPG 算法。

3 个算法在算例 kp2、kp4、kp6、kp8、kp10 上最快收敛时间的对比如图 10 所示，其中算例涵盖了由

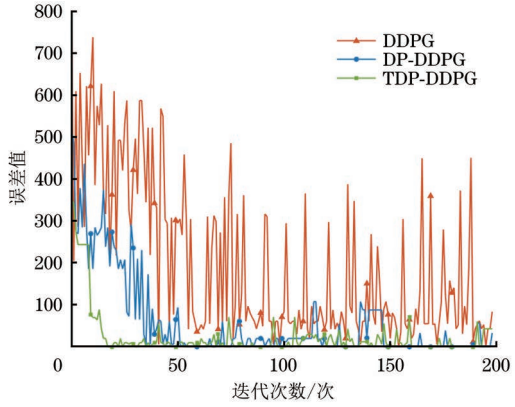

图 9 不同算法在 kp8 上的收敛对比图

Fig. 9 Comparison plot of different algorithms on kp8

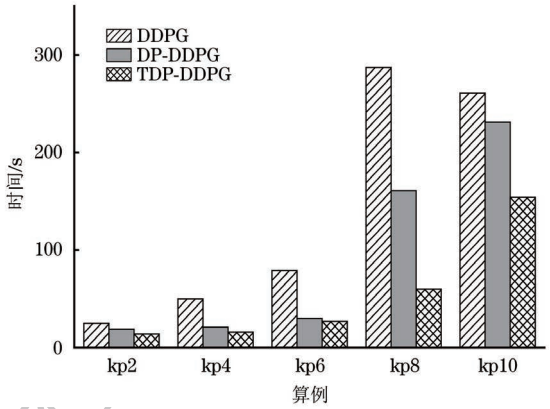

图 10 不同算法的收敛时间对比 1

Fig. 10 Comparison 1 of convergence times of different algorithms

低到高维度。本文的最快收敛时间都表示为第一次收敛到最优值的截止时间，若无法收敛到最优值，则取整个过程中的最大值。可以看出，TDP-DDPG 算法在首次达到最优值的收敛时间整体上是最快的，DP-DDPG 算法次之，而原始 DDPG 算法在某些算例不能收敛到最优值的情况下收敛时间最长。在求解时间上，与 TDP-DDPG 算法精度相近的 CBA 算法在求解低维度算例时可以在 50 次迭代内求得最优解，在求解中维度算例时在 200 次迭代内可以求得最优解，而在高维度算例 kp10 中无法求得最优解。由于使用 DRL 算法进行训练，且设备有限，TDP-DDPG 算法的每次迭代约为 2 s，相比于智能算法较长。但在迭代次数上 TDP-DDPG 算法在求解低维度算例时可以在 50 次迭代内求得最优解，在求解中高维度算例时在 150 次迭代内可以求得最优解，可以以更少的迭代次数求得最优解，迭代效率高。

综合来看，TDP-DDPG 算法在收敛速度和收敛精度上都是最优的，在改进策略的帮助下不仅可以在局部最优的情况下更快逃逸出来，而且可以收敛到最优值，在收敛速度和准确度上效果最好。

5.2 TDP-DDPG 算法在测试集 2 上的性能对比

实验在测试集 2 上依然将每个算例在训练时分为 10 组，每组分别训练迭代 200 次，并且单独保存训练结果。

选用 BLSO、HBA、ACPSO 和 BBA 算法进行对比，实验数据如表 6 所示。TDP-DPPG 算法和 DP-DPPG 算法在测试集 2 上的测试结果如表 7 所示。由表 6 可知，原始 DDPG 算法仅能在低维度的 3 个算例中达到最优，TDP-DDPG 算法和 DP-DDPG 算法仍可以在测试集 2 中的所有算例中都能达到最优。TDP-DDPG 算法在测试集 2 中的 5 个高维度算例中表现优异，说明改进策略在较高物品维度的算例中具有普遍适用性，能稳定地适用于不同类型算例。由表 7 可知，总体上两个算法测试结果中均值解和标准差的比较与测试集 1 相比无太大差距，但在规模稍大的 kp15、kp16 算例中，DP-DDPG 算法的均值解略差于 TDP-DDPG 算法，但也几乎接近最优均值。TDP-DDPG 算法在成功率和标准差上优于 DP-DDPG 算法，说明融入 Transformer 模块后算法的稳定性得到了提升。

表 6 TDP/DP-DDPG、DDPG 算法在测试集 2 上的实验数据对比

Table 6 Comparison of experimental data of TDP/DP-DDPG, DDPG algorithms on test set 2

算例	问题规模	理想最优解	BLSO	ACPSO	HBA	BBA	DDPG	DP-DDPG	TDP-DDPG
kp11	10	295	**295**	**295**	**295**	**295**	**295**	**295**	**295**
kp12	20	1 024	**1 024**	**1 024**	**1 024**	**1 024**	**1 024**	**1 024**	**1 024**
kp13	20	1 042	**1 042**	**1 042**	**1 042**	**1 042**	**1 042**	**1 042**	**1 042**
kp14	50	4 882	**4 882**	**4 882**	**4 882**	**4 882**	4 844	**4 882**	**4 882**
kp15	100	15 170	**15 170**	**15 170**	**15 170**	**15 170**	15 111	**15 170**	**15 170**
kp16	100	26 559	**26 559**	**26 559**	**26 559**	**26 559**	26 444	**26 559**	**26 559**
kp17	100	2 660	**2 660**	**2 660**	**2 660**	**2 660**	2 631	**2 660**	**2 660**
kp18	100	4 143	**4 143**	**4 143**	**4 143**	**4 143**	4 122	**4 143**	**4 143**
kp19	100	4 987	**4 987**	4 986	**4 987**	**4 987**	4 948	**4 987**	**4 987**

表 7 TDP/DP-DDPG 算法在测试集 2 上的测试结果

Table 7 Results of TDP/DP-DDPG algorithm on test set 2

算例	问题规模	理想最优解	算法	最优解	最差解	均值解	标准差	成功率/%
kp11	10	295	TDP-DDPG	295	295	295	0	100
			DP-DDPG	295	295	295	0	100
kp12	20	1 024	TDP-DDPG	1 024	1 024	1 024	0	100
			DP-DDPG	1 024	1 024	1 024	0	100
kp13	20	1 042	TDP-DDPG	1 042	1 042	1 042	0	100
			DP-DDPG	1 042	1 042	1 042	0	100
kp14	50	4 882	TDP-DDPG	4 882	4 882	4 882	0	100
			DP-DDPG	4 882	4 882	4 882	0	100
kp15	100	15 170	TDP-DDPG	15 170	15 170	15 170	0	100
			DP-DDPG	15 170	15 164	15 169.4	1.8	90
kp16	100	26 559	TDP-DDPG	26 559	26 559	26 559	0	100
			DP-DDPG	26 559	26 553	26 558.4	1.8	90
kp17	100	2 660	TDP-DDPG	2 660	2 660	2 660	0	100
			DP-DDPG	2 660	2 660	2 660	0	100
kp18	100	4 143	TDP-DDPG	4 143	4 143	4 143	0	100
			DP-DDPG	4 143	4 143	4 143	0	100
kp19	100	4 987	TDP-DDPG	4 987	4 987	4 987	0	100
			DP-DDPG	4 987	4 987	4 987	0	100

为了解 8 个算法之间对比的差异性，在置信水平为 0.05 的条件下，使用非参数 Wilcoxon 秩和检验，结果如表 8 所示。可以看出，TDP-DDPG 算法和 DP-DDPG 算法的收敛精度与 BLSO、HBA、DPSO、BBA 算法相同，对 ACPSO 有略微的优势，但是对原始 DDPG 算法具有显著优势。

表 8 TDP-DDPG 与 6 个对比算法在测试集 2 上的秩和检验测试结果

Table 8 Test results of rank sum test between TDP-DDPG and 6 comparison algorithms on test set 2

算法对比	Better	Equal	Worse	p 值
TDP-DDPG vs DDPG	6	3	0	0.027 7
TDP-DDPG vs DP-DDPG	0	9	0	1.000 0
TDP-DDPG vs BLSO	0	9	0	1.000 0
TDP-DDPG vs ACPSO	1	8	0	0.317 3
TDP-DDPG vs HBA	0	9	0	1.000 0
TDP-DDPG vs BBA	0	9	0	1.000 0

精度相近的算法在各个算例上的收敛时间对比如图 11 所示，可以发现 TDP-DDPG 算法在 4 个中低维度算例中 3 个表现优于 ACPSO 算法，在 5 个高维度算例中 2 个优于 ACPSO 算法，相比于 BLSO、HBA、BBA 算法整体上收较慢。

各算法的成功率对比如表 9 所示，其中 TDP-DDPG 算法与 BLSO 算法可以在所有算例中 100%求得最优解。与其他 3 种算法相比，TDP-DDPG 算法则具有优势，特别是与收敛时间互有胜负的 ACPSO 算法相比成功率有很大提升，表现更加稳定。

3 个算法在算例 kp15 上的数据分析图如图 12、图 13 所示。可以看出：TDP-DDPG 算法仍能以较少迭代次数先于 DP-DDPG 算法达到误差 0，且稳定波动，说明改进策略可以较好地适应规模变大的问题，且收敛精度不受影响；原始 DDPG 算法在收敛时间上受到物品数量维度和规模增大的影响比测

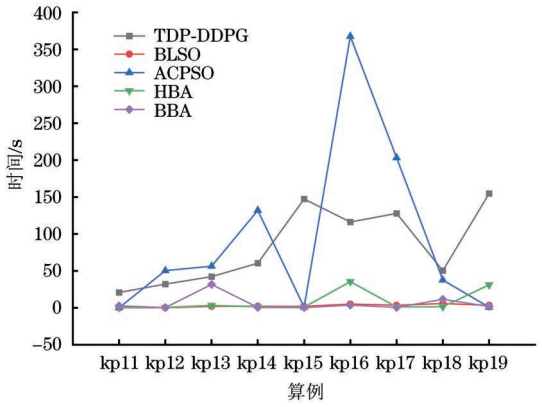

图 11 不同算法的收敛时间对比 2

Fig. 11 Comparison 2 of convergence times of different algorithms

表 9 算法成功率对比

Table 9 Comparison of algorithms in terms of success rate %

算例	TDP-DDPG	BLSO	ACPSO	HBA	BBA
kp11	100	100	100.00	100.00	96.67
kp12	100	100	53.33	100.00	100.00
kp13	100	100	46.67	100.00	60.00
kp14	100	100	16.67	100.00	100.00
kp15	100	100	100.00	100.00	100.00
kp16	100	100	3.33	46.67	100.00
kp17	100	100	43.33	100.00	100.00
kp18	100	100	100.00	100.00	96.67
kp19	100	100	100.00	50.00	100.00
AVG	100	100	62.59	88.52	94.82

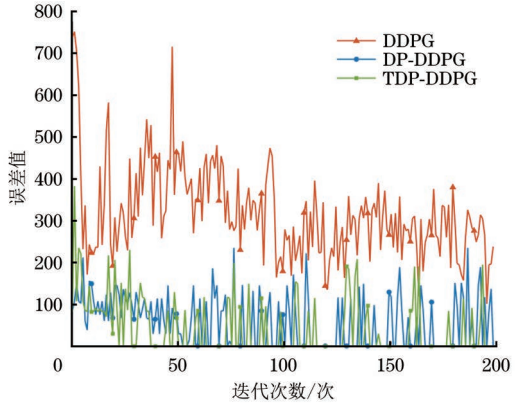

图 12 不同算法在 kp15 上的收敛对比图

Fig. 12 Comparison plot of different algorithms on kp15

试集 1 变得更大；TDP-DDPG 算法依然可以最快收敛到误差 0；而 DP-DDPG 算法和测试集 1 相同，收敛到误差 0 的速度稍逊于 TDP-DDPG 算法。

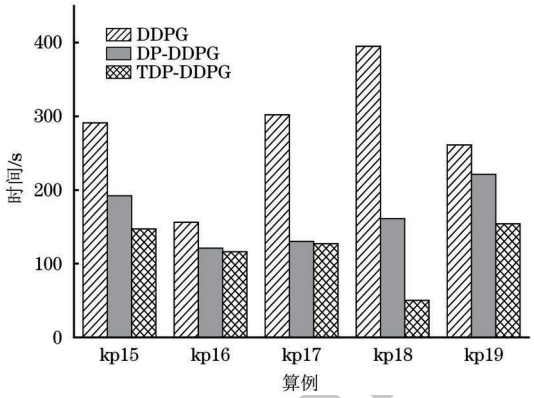

图 13 不同算法的收敛时间对比 3

Fig. 13 Comparison 3 of convergence times of different algorithms

5.3 TDP-DDPG 算法在测试集 3 上的性能对比

实验在测试集 3 上将每个算例在训练时分为 10 组，每组分别训练迭代 200 次，在 1 000 维的物品维度下每组分别训练迭代 500 次并且单独保存训练结果。引入基于转移函数的二元爬行搜索算法(BinRSA)[35]、基于 PSO 算法学习机制的改进差分进化算法——二分学习差分进化算法(BLDE)[46]、二分二元差分进化(DBDE)[47]、二进制帝国竞争算法(BICA)[26]、新型二元差分进化算法(Nbin-DE)[48]进行实验对比，如表 10 所示。

表 10 TDP/DP-DDPG、DDPG 算法在测试集 3 上的实验数据对比

Table 10 Comparison of experimental data of TDP/DP-DDPG, DDPG algorithms on test set 3

算例	问题规模	最优解	BLDE	DBDE	BinRSA	Nbin-DE	BICA	DDPG	DP-DDPG	TDP-DDPG
kp20	100	1 807	**1 807**	**1 807**	**1 807**	**1 807**	**1 807**	**1 737**	**1 807**	**1 807**
kp21	100	659	658	**659**	**659**	**659**	**659**	647	**659**	**659**
kp22	100	813	812	**813**	**813**	**813**	**813**	808	**813**	**813**
kp23	100	493	**493**	**493**	**493**	**493**	**493**	**493**	**493**	**493**
kp24	200	3 403	3 401	**3 403**	**3 403**	**3 403**	**3 403**	3 286	**3 403**	**3 403**
kp25	200	1 332	1 329	**1 332**	**1 332**	**1 332**	**1 332**	1 320	**1 332**	**1332**
kp26	200	1 631	1 626	**1 631**	**1 631**	**1 631**	**1 631**	1 621	**1 631**	**1 631**
kp27	200	1 001	**1 001**	**1 001**	**1 001**	**1 001**	**1 001**	**1 001**	**1 001**	**1 001**
kp28	300	5 444	5 441	**5 444**	5 443	**5 444**	**5 444**	5 432	**5 444**	**5 444**
kp29	300	1 963	1 961	**1 963**	**1 963**	**1 963**	**1 963**	1 956	**1 963**	**1 963**
kp30	300	2 433	2 426	**2 433**	**2 433**	**2 433**	**2 433**	2 391	**2 433**	**2 433**
kp31	300	1 523	**1 523**	**1 523**	**1 523**	**1 523**	**1 523**	**1 523**	**1 523**	**1 523**
kp32	500	9 495	9 484	**9 495**	**9 495**	**9 495**	**9 495**	9 331	**9 495**	**9 495**
kp33	500	3 250	3 247	3 247	**3 250**	**3 250**	**3 250**	3 139	**3 250**	**3 250**
kp34	500	4 078	4 051	4 070	**4 078**	**4 078**	**4 078**	3 846	**4 078**	**4 078**
kp35	500	2 518	**2 518**	**2 518**	**2 518**	**2 518**	**2 518**	**2 518**	**2 518**	**2 518**
kp36	1 000	18 844	18 492	18 843	**18 844**	**18 844**	**18 844**	18 705	**18 843**	**18 844**
kp37	1 000	6 482	6 458	6 463	**6 482**	**6 482**	**6 482**	6 383	6 480	**6 482**
kp38	1 000	8 228	8 073	8 109	**8 228**	**8 228**	**8 228**	8 065	**8 228**	**8 228**
kp39	1 000	5 068	**5 068**	**5 068**	**5 068**	**5 068**	**5 068**	**5 068**	**5 068**	**5 068**

测试集 3 内 100～1 000 问题规模的算例中，每一种问题规模的 4 个算例按照数据类型依次排列为无相关、弱相关、强相关、完全相关。其中完全相关的算例寻优效率最高，可以在第 0 代或者第 1 代达到最优，并且不受问题规模增大的影响。TDP-DPPG 算法和 DP-DPPG 算法在测试集 3 上的测试结果如表 11 所示。

表 11 TDP/DP-DDPG 算法在测试集 3 上的测试结果

Table 11 Results of TDP/DP-DDPG algorithm on test set 3

算例	问题规模	理想最优解	算法	最优解	最差解	均值解	标准差	成功率/%
kp20	100	1 807	TDP-DDPG	1 807	1 807	1 807	0	100
			DP-DDPG	1 807	1 807	1 807	0	100
kp21	100	659	TDP-DDPG	659	659	659	0	100
			DP-DDPG	659	659	659	0	100
kp22	100	813	TDP-DDPG	813	813	813	0	100
			DP-DDPG	813	813	813	0	100
kp23	100	493	TDP-DDPG	493	493	493	0	100
			DP-DDPG	493	493	493	0	100
kp24	200	3 403	TDP-DDPG	3 403	3 403	3 403	0	100
			DP-DDPG	3 403	3 403	3 403	0	100
kp25	200	1 332	TDP-DDPG	1 332	1 332	1 332	0	100
			DP-DDPG	1 332	1 332	1 332	0	100
kp26	200	1 631	TDP-DDPG	1 631	1 631	1 631	0	100
			DP-DDPG	1 631	1 631	1 631	0	100
kp27	200	1 001	TDP-DDPG	1 001	1 001	1 001	0	100
			DP-DDPG	1 001	1 001	1 001	0	100
kp28	300	5 444	TDP-DDPG	5 444	5 444	5 444	0	100
			DP-DDPG	5 444	5 444	5 444	0	100
kp29	300	1 963	TDP-DDPG	1 963	1 963	1 963	0	100
			DP-DDPG	1 963	1 963	1 963	0	100
kp30	300	2 433	TDP-DDPG	2 433	2 433	2 433	0	100
			DP-DDPG	2 433	2 433	2 433	0	100
kp31	300	1 523	TDP-DDPG	1 523	1 523	1 523	0	100
			DP-DDPG	1 523	1 523	1 523	0	100
kp32	500	9 495	TDP-DDPG	9 495	9 495	9 495	0	100
			DP-DDPG	9 495	9 495	9 495	0	100
kp33	500	3 250	TDP-DDPG	3 250	3 250	3 250	0	100
			DP-DDPG	3 250	3 247	3 250	0	100
kp34	500	4 078	TDP-DDPG	4 078	4 078	4 078	0	100
			DP-DDPG	4 078	4 078	4 078	0	100
kp35	500	2 518	TDP-DDPG	2 518	2 518	2 518	0	100
			DP-DDPG	2 518	2 518	2 518	0	100
kp36	1 000	18 844	TDP-DDPG	18 844	18 844	18 844	0	100
			DP-DDPG	18 843	18 832	18 839.9	4.5	0
kp37	1 000	6 482	TDP-DDPG	6 482	6 482	6 482	0	100
			DP-DDPG	6 480	6 473	6 478.6	2.8	0
kp38	1 000	8 228	TDP-DDPG	8 228	8 228	8 228	0	100
			DP-DDPG	8 228	8 226	8 227.2	0.9	80
kp39	1 000	5 068	TDP-DDPG	5 068	5 068	5 068	0	100
			DP-DDPG	5 068	5 068	5 068	0	100

由表 10、表 11 可以看出，当物品的数量维度持续增高时，原始 DDPG 算法仅能在完全相关的 5 个算例中找到最优值。而加入 DRM、DPM 两个优化策略后的 DP-DDPG 算法依然不受物品数量维度增

高的影响，可以高效寻优，在 500 个物品数量及以下的所有算例中均能 100%求得最优解，在 1 000 个物品数量的算例中两个求得最优解，两个求得次优解。这说明两个优化策略对于算法的性能提升起到了很大的效果。TDP-DDPG 算法在融入 Transformer 模块后，在成功率和求解精度上优于 DP-DDPG 算法，在更高物品维度的算例中发挥了稳定的高效性，与 BICA、Nbin-DE 算法在求解精度上效果不相上下，在所有对比算法中处于最优。

为了解 7 个算法之间对比的差异性，同样在置信水平为 0.05 的条件下，使用非参数 Wilcoxon 秩和检验，结果如表 12 所示。可以看出，TDP-DDPG 算法依然对 DDPG、BLDE 算法具有显著优势，对于 DP-DDPG、DBDE 算法在高物品数量维度的算例中有较大优势。说明随着问题规模和数量维度的增高，改进策略可以很好地适应并发挥作用。TDP-DDPG、BICA、Nbin-DE 算法在寻优精度上持平，都可以在所有算例中找到最优值。

表 12 TDP-DDPG 与 7 个对比算法在测试集 3 上的秩和检验测试结果

Table 12 Test results of rank sum test between TDP-DDPG and 7 comparison algorithms on test set 3

算法对比	Better	Equal	Worse	p 值
TDP-DDPG vs DDPG	15	5	0	0.000 6
TDP-DDPG vs DP-DDPG	2	18	0	0.179 7
TDP-DDPG vs BLDE	14	6	0	0.000 9
TDP-DDPG vs DBDE	5	15	0	0.043 1
TDP-DDPG vs BICA	0	20	0	1.000 0
TDP-DDPG vs BinRSA	1	19	0	0.317 3
TDP-DDPG vs Nbin-DE	0	20	0	1.000 0

为检验以 DDPG 算法作为实验的基础算法是否具有性能优势，将实验的所有改进策略增加到另一个深度强化学习算法 DQN 上组成 DQN-baseline 算法与之对比。由于两个算法都能在迭代次数为 0 或 1 次时求得完全相关算例的最优值，故不将完全相关算例作为本次实验的对比对象。在同样迭代 200 次的前提下，两个算法的运行时间很接近，在训练时间的比较上没有明显差距。

表 13 为两个算法在测试集 3 上的精度对比，可以看出当算例的维度为 100 时，DQN-baseline 算法与 TDP-DDPG 算法在 kp20 上都能求得最优值，在另外两个 100 维度的算例上 DQN-baseline 算法无法求得最优值，但是与 TDP-DDPG 算法相比差距不大。在 300～1 000 个算例的问题中两个算法在求解精度上开始具有明显差距。图 14 表示两个算法在实验结果差值上的对比，可以看出随着算例维度的增高，两个算法之间的差值整体上呈上升趋势，且在规模增大的 kp36 算例上差值最为明显。这说明 TDP-DDPG 算法相比 DQN-baseline 算法能够更好地适应维度和规模增大的问题，保持高效的求解效率和精度。

通过测试集 3 来验证 3 个 DRL 算法的性能。3 个算法算例在 kp33 上的收敛曲线图如图 15 所示。从收敛趋势可知，原始 DDPG 算法与加入改进策略的两个算法差距越来越明显。从迭代次数可以看到，TDP-DDPG 算法与 DP-DDPG 算法第一次迭代到最优值的次数相差了约 100，这相比于物品数量维度较低的前两个测试集也在增大。综合来看，改进策略 DRM、DPM 的加入仍然使算法可以高效精确地收敛，而融入 Transformer 模块后使算法可以更快更稳定地收敛到最优值。

表 13 TDP-DDPG 算法与 DQN-baseline 算法的精度对比

Table 13 Comparison of the accuracy of TDP-DDPG algorithm and DQN-baseline algorithm

算例	问题规模	最优解	DQN-baseline	TDP-DDPG
kp20	100	1 807	1 807	1 807
kp21	100	659	654	659
kp22	100	813	783	813
kp24	200	3 403	3 395	3 403
kp25	200	1 332	1 299	1 332
kp26	200	1 631	1 591	1 631
kp28	300	5 444	5 323	5 444
kp29	300	1 963	1 916	1 963
kp30	300	2 433	2 401	2 433
kp32	500	9 495	9 266	9 495
kp33	500	3 250	3 159	3 250
kp34	500	4 078	3 968	4 078
kp36	1 000	18 844	17 786	18 844
kp37	1 000	6 482	6 325	6 482
kp38	1 000	8 228	7 976	8 228

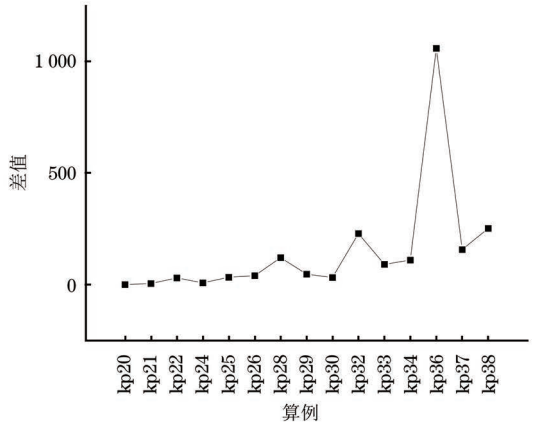

图 14 TDP-DDPG 算法与 DQN-baseline 算法的差值分析

Fig. 14 Difference analysis between TDP-DDPG algorithm and DQN-baseline algorithm

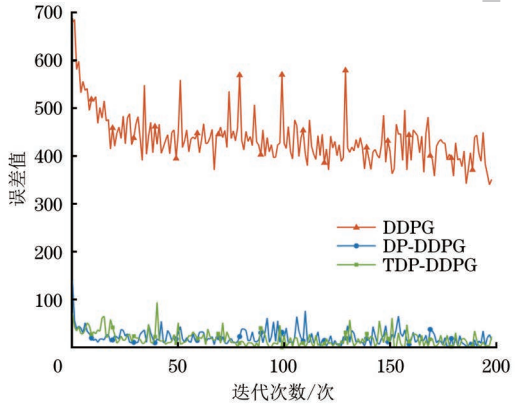

图 15 不同算法在 kp33 上的收敛对比图

Fig. 15 Comparison plot of different algorithms on kp33

3 个算法在算例 kp21、kp25、kp29、kp33、kp37 最快收敛时间对比如图 16 所示。某些算例出现 DP-DDPG算法收敛比TDP-DDPG算法收敛时间快的情况，首先融入 Transformer 模块后数据传入处理时更为复杂，每迭代一次所用时间要比未加入 Transformer 模块时快；其次 DRL 选择具有一定的随机性和探索能力，其偶然性也可能导致这种情况发生。

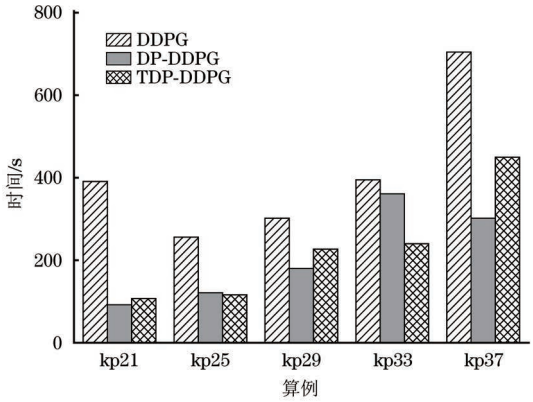

图 16 不同算法的收敛时间对比 4

Fig. 16 Comparison 4 of convergence times of different algorithms

5.4 TDP-DDPG 算法在测试集 4 上的性能对比

实验在大规模测试集 4 上依然将每个算例在训练时分为 10 组，每组分别训练迭代 300 次并且单独保存训练结果。

两个改进算法在测试集 4 上的测试结果如表 14 所示。在大规模算例下，TDP-DDPG 算法在 800 物品数量维度中虽然成功率稍有下降，但仍可以较为稳定地收敛到最优值，在 1 000 物品数量维度中可以稳定收敛到非常接近最优值的次优值。DP-DDPG 算法除了在 kp40、kp41 可以收敛到最优值外，在 kp42～kp45 算例中都可以稳定收敛到一个较优的值，其稳定性相较于 TDP-DDPG 算法差距较小，但收敛精度相较于 TDP-DDPG 算法总体上具有较大差距。

表 14 TDP/DP-DDPG 算法在测试集 4 上的测试结果

Table 14 Results of TDP/DP-DDPG algorithm on test set 4

算例	问题规模	理想最优解	算法	最优解	最差解	均值解	标准差	成功率/%
kp40	800	40 686	TDP-DDPG	40 686	40 677	40 683.3	3.8	60
			DP-DDPG	40 686	40 672	40 681	5.0	40
kp41	800	35 069	TDP-DDPG	35 069	35 057	35 066.9	4.2	80
			DP-DDPG	35 069	35 043	35 058.8	11.2	50
kp42	800	40 167	TDP-DDPG	40 167	40 157	40 165	3.1	80
			DP-DDPG	40 161	40 154	40 159	2.8	0
kp43	1 000	50 592	TDP-DDPG	50 590	50 586	50 588.6	1.5	0
			DP-DDPG	50 576	50 571	50 575	1.6	0
kp44	1 000	43 786	TDP-DDPG	43 775	43 744	43 769.4	9.8	0
			DP-DDPG	43 740	43 717	43 735.7	7.5	0
kp45	1 000	49 443	TDP-DDPG	49 442	49 438	49 440	1.8	0
			DP-DDPG	49 435	49 426	49 430	4.3	0

TDP-DDPG 算法在算例 kp45 下的收敛曲线图如图 17 所示。可以看出算法可以很快收敛到误差值 0 的周围并且稳定波动。在整个收敛过程中算法进行若干次探索，在每次探索后均可以再

次收敛。结合表 14 来看，改进策略 DRM、DPM 虽然在大规模算例中受到一些影响，收敛精度虽有所下降，但仍可以稳定收敛到较为理想的值。而 Transformer 模块的融入可以在一定程度上缓解规模变大所带来的影响，提升算法的稳定性和准确率。

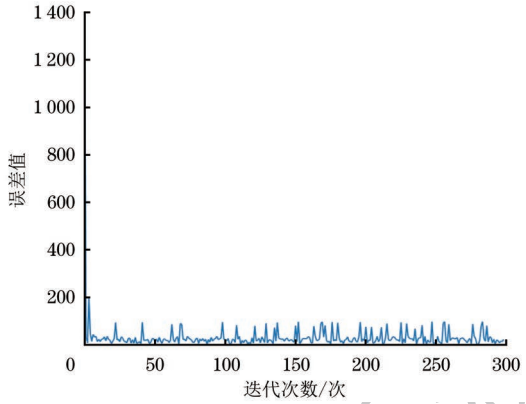

图 17 TDP-DDPG 算法在 kp45 上的收敛对比图

Fig. 17 Comparison plot of TDP-DDPG algorithms on kp45

5.5 BPD-DDPG 算法性能分析

为更加贴合 HMKP 的抽象数据特征，选用测试集 3 中的 2 或 3 个算例自由组合成符合 BPD-DDPG 算法结构的 5 个组合算例(kp46～kp50)，组合数量维度最低为 200，最高为 2 000。以相同的形式选用大规模测试集 6 中的 2 或 3 个算例自由组合成 5 个组合算例(kp51～kp55)，组合数量维度最低为 1 600，最高为 3 000。以 kp46～kp55 组成测试集 5 来分析 BPD-DDPG 算法的性能。

实验在测试集 5 上将每个算例在训练时分为 10 组，每组分别训练迭代 300 次，并且单独保存训练结果。各个背包内的物品对每个背包都具有一个二进制位，若物品可被装入背包则置为 1，否则置为 0。将每个背包的理想最优值与测试最优值之差 Gap_n 作为测试结果，G_{min} 表示本次实验所有背包的差值总和。用最优解、最差解、均值解、标准差、迭代次数来评估算法性能，其中迭代次数为第一次收敛到最优的次数。测试集 5 的测试结果如表 15 所示。

表 15 BPD-DDPG 算法在测试集 5 上的测试结果

Table 15 Results of BPD-DDPG algorithm on test set 5

算例	参数	最优解	最差解	均值解	标准差	迭代次数/次	成功率/%
kp46	G_{min}	0	0	0	0	21	100
	Gap_n	0, 0	0, 0	0, 0			
kp47	G_{min}	0	0	0	0	78	100
	Gap_n	0, 0	0, 0	0, 0			
kp48	G_{min}	0	1.00	0.20	0.40	93	90
	Gap_n	0, 0, 0	0, 0, 1.00	0, 0, 0.20			
kp49	G_{min}	0	0	0	0	89	100
	Gap_n	0, 0, 0	0, 0	0, 0, 0			
kp50	G_{min}	0	1.00	0.10	0.30	102	80
	Gap_n	0, 0, 0	0, 0, 1.00	0, 0, 0.10			
kp51	G_{min}	2.00	11.00	4.10	2.78	172	0
	Gap_n	0, 2.00	9.00, 6.00	1.50, 2.60			
kp52	G_{min}	11.00	42.00	27.70	7.14	96	0
	Gap_n	0, 11.00	12.00, 42.00	3.30, 24.40			
kp53	G_{min}	1.00	14.00	3.50	4.20	73	0
	Gap_n	0, 1.00	10.00, 4.00	2.20, 1.30			
kp54	G_{min}	0	31.00	9.00	8.10	157	0
	Gap_n	0, 0, 0	9.00, 12.00, 10.00	1.50, 3.10, 4.40			
kp55	G_{min}	14.00	0.00	32.10	7.80	193	0
	Gap_n	2.00, 11.00, 1.00	6.00, 42.00, 4.00	3.10, 27.70, 1.30			

采用 Gurobi[49] 求解器对测试 5 中的算例进行求解，与 BPD-DDPG 算法的测试结果进行对比，对比结果如表 16 所示。Gurobi 求解器具有求解精度高、求解速度快的特点，且求解结果十分稳定。将 BPD-DDPG 算法与 Gurobi 求解器在测试集上进行结果对比可以有效验证 BPD-DDPG 算法性能。

由表 15 和表 16 中的测试结果及对比可以看出，BPD-DDPG 算法在测试集 5 中的 10 个算例 6 个可以达到最优值，其中大规模算例占 1 个。说明算法在理想状态下依然保持了 TDP-DDPG 算法

的求解能力，改进策略在异构多背包背景下依然可以发挥作用。

表 16 BPD-DDPG 算法与 Gurobi 在测试集 5 上效果对比

Table 16 Comparison of the effectiveness of the BPD-DDPG algorithm with the Gurobi solver on test set 5

算例	BPD-DDPG		Gurobi	
	最优 G_{min}	用时/s	最优 G_{min}	用时/s
kp46	0	45.52	0	0.16
kp47	0	264.58	1	1.03
kp48	0	348.41	1	1.12
kp49	0	287.50	0	0.55
kp50	0	394.34	1	1.25
kp51	2	2 380.14	1	1.48
kp52	11	1 191.96	2	1.69
kp53	1	943.12	1	0.84
kp54	0	2 653.12	2	1.83
kp55	14	2 926.54	2	2.19
AVG(kp46～kp50)	0	268.07	0.6	0.82
AVG(kp51～kp55)	28	2 018.98	1.6	1.61

由于 HMKP 的复杂性较高，算法在求解时间上长，相较于 Gurobi 求解器差距较大。在非大规模的算例(kp46～50)中求解精度比 Gurobi 求解器高，在大规模算例(kp51～kp55)中的求解精度在个别算例上比 Gurobi 求解器高或者持平，但在稳定性上 Gurobi 求解器更突出。BPD-DDPG 算法在整个训练过程中大部分时间保持着高效率状态，在求解 HMKP 时可以适应问题特性，维持寻优能力强的特点，可以在一定程度上较好地解决 HMKP。

6 结束语

本文从 0-1 KP 出发，结合传统多 KP 的变种——HMKP，设计了符合 HMKP 特征的改进 DDPG 算法。针对原始 DDPG 算法的不足，采用 DRM 和 DPM 两个优化策略来提高算法性能，并且在此基础上通过嵌入改进 Transformer 模块来优化 DDPG 算法中状态和动作的策略生成，同时加入禁忌表防止重复搜索，从而提出解决 0-1 KP 的 TDP-DDPG 算法。通过 45 个算例的实验证明以及与众多启发式算法的比较，显示了 TDP-DDPG 算法具有快速、准确的寻优能力，验证了改进策略的有效性。针对 HMKP 约束性强、复杂度高的特点，基于 TDP-DDPG 算法设计了 BPD-DDPG 算法，并且在 10 个不同规模的算例上进行实验分析，最后与商业数学规划优化器 Gurobi 进行结果对比。在与 Gurobi 的对比实验中虽然时间上处于劣势，但在一些算例上求解精度更好或持平，说明 BPD-DDPG 算法可以有效解决 HMKP。未来将在复杂物流系统的港口码头异构作业空间调度场景的组合优化问题中展开实际应用探讨。

参考文献

[1] DANTZIG G B. Discrete-variable extremum problems[J]. Operations Research, 1957, 5(2): 266-288.

[2] WANG C, PAN Q K, SANG H Y, et al. A cascaded flowshop joint scheduling problem with makespan minimization: a mathematical model and shifting iterated greedy algorithm[J]. Swarm and Evolutionary Computation, 2024, 86: 101489.

[3] SONG X R, GAO S, CHEN C B, et al. A new hybrid method in global dynamic path planning of mobile robot[J]. International Journal of Computers Communications & Control, 2018, 13(6): 1032-1046.

[4] WU Y H, MENG X Y, ZHANG J R, et al. Effective LSTMs with seasonal-trend decomposition and adaptive learning and niching-based backtracking search algorithm for time series forecasting[J]. Expert Systems with Applications, 2024, 236: 121202.

[5] ZHANG S, LIU S Y. A discrete improved artificial bee colony algorithm for 0-1 knapsack problem[J]. IEEE Access, 2019, 7: 104982-104991.

[6] ÖZTÜRK S, AHMAD R, AKHTAR N. Variants of artificial bee colony algorithm and its applications in medical image processing[J]. Applied Soft Computing, 2020, 97: 106799.

[7] ABDEL-BASSET M, EL-SHAHAT D, SANGAIAH A K. A modified nature inspired meta-heuristic whale optimization algorithm for solving 0-1 knapsack problem[J]. International Journal of Machine Learning and Cybernetics, 2019, 10(3): 495-514.

[8] GHAREHCHOPOGH F S, GHOLIZADEH H. A comprehensive survey: whale optimization algorithm and its applications[J]. Swarm and Evolutionary Computation, 2019, 48: 1-24.

[9] WU L S, YOU X M, LIU S. Multi-ant colony algorithm based on cooperative game and dynamic path tracking[J]. Computer Networks, 2023, 237: 110077.

[10] WU C J, ZHOU S J, XIAO L C. Dynamic path planning based on improved ant colony algorithm in traffic congestion[J]. IEEE Access, 2020, 8: 180773-180783.

[11] YANG Z L. Competing leaders grey wolf optimizer and its application for training multi-layer perceptron classifier[J]. Expert Systems with Applications, 2024, 239: 122349.

[12] 刘志强，何丽，袁亮，等. 采用改进灰狼算法的移动机器人路径规划[J]. 西安交通大学学报，2022，56(10)：49-60.
LIU Z, HE L, YUAN L, et al. Path planning of mobile robot based on TGWO algorithm[J]. Journal of Xi'an Jiaotong University, 2022, 56: 49-60. (in Chinese)

[13] 杨艳，刘生建，周永权. 贪心二进制狮群优化算法求解多维背包问题[J]. 计算机应用，2020，40(5)：1291-1294.
YANG Y, LIU S J, ZHOU Y Q. Greedy binary lion swarm optimization algorithm for solving multidimensional knapsack problem[J]. Journal of Computer Applications, 2020, 40(5): 1291-1294. (in Chinese)

[14] LIU J F, LI D F, WU Y, et al. Lion swarm optimization algorithm for comparative study with application to optimal dispatch of cascade hydropower stations[J]. Applied Soft Computing, 2020, 87: 105974.

[15] WANG Q, HAO Y S, ZHANG J W. Generative inverse

reinforcement learning for learning 2-opt heuristics without extrinsic rewards in routing problems[J]. Journal of King Saud University-Computer and Information Sciences, 2023, 35(9): 101787.
[16] BELLO I, PHAM H, LE Q V, et al. Neural combinatorial optimization with reinforcement learning[EB/OL]. [2024-08-01]. https://arxiv.org/abs/1611.09940.
[17] KOOL W, VAN HOOF WELLING M. Attention, learn to solve routing problems![EB/OL]. [2024-08-01]. https://arxiv.org/abs/1803.08475.1.
[18] HU W X, ISHIHARA H, CHEN C H, et al. Deep reinforcement learning two-way transit signal priority algorithm for optimizing headway adherence and speed[J]. IEEE Transactions on Intelligent Transportation Systems, 2023, 24(8): 7920-7931.
[19] WANG Y, CHEN Z B. Dynamic graph Conv-LSTM model with dynamic positional encoding for the large-scale traveling salesman problem [J]. Mathematical Biosciences and Engineering, 2022, 19(10): 9730-9748.
[20] WANG Q, HAO Y S. Routing optimization with Monte Carlo tree search-based multi-agent reinforcement learning [J]. Applied Intelligence, 2023, 53(21): 25881-25896.
[21] KHALIL E, DAI H, ZHANG Y, et al. Learning combinatorial optimization algorithms over graphs[C]// Proceedings of the 31st International Conference on Neural Information Processing System. New York, USA: ACM Press, 2017: 6351-636.
[22] SUI J, DING S, LIU R, et al. Learning 3-opt heuristics for traveling salesman problem via deep reinforcement learning[C]// Proceedings of Asian Conference on Machine Learning. [S. l.]: PMLR, 2021: 1301-1316.
[23] 王景熠, 朱予涵, 周茹平, 等. 基于动态粒子群优化的 X 结构 Steiner 最小树算法[J]. 计算机工程, 2024, 50(9): 226-234.
WANG J Y, ZHU Y H, ZHOU R P, et al. X-architecture Steiner minimum tree algorithm based on dynamic particle swarm optimization [J]. Computer Engineering, 2024, 50(9): 226-234. (in Chinese)
[24] 葛非, 闵珊, 邱含, 等. 求解时间依赖型绿色车辆路径问题的算法研究[J]. 计算机工程, 2024, 50(4): 1-10.
GE F, MIN S, QIU H, et al. Research on an algorithm for solving time-dependent green vehicle routing problem [J]. Computer Engineering, 2024, 50(4): 1-10. (in Chinese)
[25] 陈思远, 林丕源, 黄沛杰. 指针网络改进遗传算法求解旅行商问题[J]. 计算机工程与应用, 2020, 56(19): 231-236.
CHEN S Y, LIN P Y, HUANG P J. Pointer network improved genetic algorithm for solving traveling salesmen problem[J]. Computer Engineering and Applications, 2020, 56(19): 231-236. (in Chinese)
[26] LI B J, WU G H, HE Y M, et al. An overview and experimental study of learning-based optimization algorithms for the vehicle routing problem [J]. CAA Journal of Automatica Sinica, 2022, 9(7): 1115-1138.
[27] LEE E S, PARK J H, KIM M Y, et al. High efficiency integrated transformer design in DAB converters for solid-state transformers [J]. IEEE Transactions on Vehicular Technology, 2022, 71(7): 7147-7160.
[28] 朱凯, 李理, 张彤, 等. 基于 Transformer 的多阶段运动模糊图像修复网络[J]. 计算机工程, 2024, 50(9): 276-285.
ZHU K, LI L, ZHANG T, et al. Multi-stage motion blur image restoration network based on Transformer [J]. Computer Engineering, 2024, 50(9): 276-285. (in Chinese)
[29] 李斌, 唐志斌. 面向异构多背包问题的多级二进制帝国竞争算法[J]. 计算机应用, 2023, 43(9): 2855-2867.
LI B, TANG Z B. Multiple level binary imperialist competitive algorithm for solving heterogeneous multiple knapsack problem [J]. Journal of Computer Applications, 2023, 43(9): 2855-2867. (in Chinese)
[30] 李斌, 李文锋. 基于 MAS 的集装箱码头物流系统协同生产调度体系[J]. 计算机集成制造系统, 2011, 17(11): 2502-2513.
LI B, LI W F. Container terminal logistics systems collaborative scheduling based on multi-agent systems [J]. Computer Integrated Manufacturing Systems, 2011, 17(11): 2502-2513. (in Chinese)
[31] ZHENG K C, JIA X L, CHI K K, et al. DDPG-based joint time and energy management in ambient backscatter-assisted hybrid underlay CRNs [J]. IEEE Transactions on Communications, 2022, 71(1): 441-456.
[32] SYAVASYA C V S R, MUDDANA A L. Optimization of autonomous vehicle speed control mechanisms using hybrid DDPG-SHAP-DRL-stochastic algorithm [J]. Advances in Engineering Software, 2022, 173: 103245.
[33] XIAO H P, FU L J, SHANG C Y, et al. Ship energy scheduling with DQN-CE algorithm combining bi-directional LSTM and attention mechanism[J]. Applied Energy, 2023, 347: 121378.
[34] ZHANG C W, ZHENG K J, TIAN Y, et al. Advertising impression resource allocation strategy with multi-level budget constraint DQN in real-time bidding [J]. Neurocomputing, 2022, 488: 647-656.
[35] ERVURAL B, HAKLI H. A binary reptile search algorithm based on transfer functions with a new stochastic repair method for 0-1 knapsack problems [J]. Computers & Industrial Engineering, 2023, 178: 109080.
[36] 陈桢, 钟一文, 林娟. 求解 0-1 背包问题的混合贪婪遗传算法[J]. 计算机应用, 2021, 41(1):87-94.
CHEN Z, ZHONG Y W, LIN J. Hybrid greedy genetic algorithm for solving 0-1 knapsack problem[J]. Journal of Computer Applications, 2021, 41(1): 87-94. (in Chinese)
[37] 刘生建, 杨艳, 周永权. 求解 0-1 背包问题的二进制狮群算法[J]. 计算机工程与科学, 2019, 41(11):2079-2087.
LIU S J, YANG Y, ZHOU Y Q. A binary lion warm algorithm for solving 0-1 knapsack problem[J]. Computer Engineering & Science, 2019, 41(11): 2079-2087. (in Chinese)
[38] 罗亚波, 滕红玺. 求解 0-1 背包问题的牵制平衡算法[J]. 工业工程, 2023, 26(3):116-123.
LUO Y B, TENG H X. An interdependence balance algorithm for solving 0-1 knapsack problems[J]. Industrial Engineering Journal, 2023, 26(3): 116. (in Chinese)
[39] 汤飞, 何永义. 基于离散二进制粒子群-模拟退火算法求解 0-1 背包问题[J]. 工业控制计算机, 2021, 34(5): 83-84, 86.
TANG F, HE Y Y. Solving 0-1 knapsack problem based on particle binary swarm-simulated annealing algorithm [J]. Industrial Control Computer, 2021, 34(5): 83-84, 86. (in Chinese)
[40] KULKARNI A J, SHABIR H. Solving 0-1 knapsack problem using cohort intelligence algorithm[J]. International Journal of Machine Learning and Cybernetics, 2016, 7(3): 427-441.
[41] ZHOU Y Q, BAO Z F, LUO Q F, et al. A complex-valued encoding wind driven optimization for the 0-1 knapsack problem[J]. Applied Intelligence, 2017, 46(3): 684-702.
[42] ZHOU Y Q, LI L L, MA M Z. A complex-valued encoding bat algorithm for solving 0-1 knapsack problem[J]. Neural Processing Letters, 2016, 44(2): 407-430.
[43] 万晓琼, 张惠珍. 求解 0-1 背包问题的混合蝙蝠算法[J]. 计算机应用研究, 2019, 36(9): 2579-2583.

WAN X Q, ZHANG H Z. Hybrid bat algorithm for solving 0-1 knapsack problem [J]. Application Research of Computers, 2019, 36(9): 2579-2583. (in Chinese)

[44] 李枝勇, 马良, 张惠珍. 求解 0/1 背包问题的自适应元胞粒子群算法[J]. 计算机工程, 2014, 40(10): 198-203.
LI Z Y, MA L, ZHANG H Z. Adaptive cellular particle swarm algorithm for solving 0/1 knapsack problem [J]. Computer Engineering, 2014, 40(10): 198-203. (in Chinese)

[45] MIRJALILI S, MIRJALILI S M, YANG X S. Binary bat algorithm[J]. Neural Computing and Applications, 2014, 25(3): 663-681.

[46] CHEN Y, XIE W C, ZOU X F. A binary differential evolution algorithm learning from explored solutions [J]. Neurocomputing, 2015, 149: 1038-1047.

[47] PENG H, WU Z J, SHAO P, et al. Dichotomous binary differential evolution for knapsack problems [J]. Mathematical Problems in Engineering, 2016, 2016: 5732489.

[48] ALI I M, ESSAM D, KASMARIK K. Novel binary differential evolution algorithm for knapsack problems[J]. Information Sciences, 2021, 542: 177-194.

[49] GUPTA S, SHU W H, ZHANG Y, et al. Differential evolution-driven traffic light scheduling for vehicle-pedestrian mixed-flow networks[J]. Knowledge-Based Systems, 2023, 274: 110636.

文字编辑 金胡考
栏目编辑 赖玉玲