基于深度强化学习的多用户计算卸载优化模型和算法

李志华* 余自立
(江南大学人工智能与计算机学院 无锡 214122)

摘 要：在移动边缘计算(MEC)密集部署场景中，边缘服务器负载的不确定性容易造成边缘服务器过载，从而导致计算卸载过程中时延和能耗显著增加。针对该问题，该文提出一种多用户计算卸载优化模型和基于深度确定性策略梯度(DDPG)的计算卸载算法。首先，考虑时延和能耗的均衡优化建立效用函数，以最大化系统效用作为优化目标，将计算卸载问题转化为混合整数非线性规划问题。然后，针对该问题状态空间大、动作空间中离散和连续型变量共存，对DDPG深度强化学习算法进行离散化改进，基于此提出一种多用户计算卸载优化方法。最后，使用该方法求解非线性规划问题。仿真实验结果表明，与已有算法相比，所提方法能有效降低边缘服务器过载概率，并具有很好的稳定性。

关键词：移动边缘计算；计算卸载；深度强化学习；资源分配
中图分类号：TN929.5;TP181 **文献标识码**：A **文章编号**：1009-5896(2024)04-1321-12
DOI: 10.11999/JEIT230445

A Multi-user Computation Offloading Optimization Model and Algorithm Based on Deep Reinforcement Learning

LI Zhihua YU Zili
(*School of Artificial Intelligence and Computer, Jiangnan University, Wuxi* 214122, *China*)

Abstract: In Mobile Edge Computing (MEC) intensive deployment scenarios, the uncertainty of edge server load can easily cause edge server overload, leading to a significant increase in delay and energy consumption during the computation offloading process. In response to this issue, a multi-user computation offloading optimization model and algorithm based on Deep Deterministic Policy Gradient (DDPG) is proposed. Firstly, considering the balance optimization of delay and energy consumption, a utility function is established to maximize system utility as the optimization objective, and the computational offloading problem is transformed into a mixed integer nonlinear programming problem. Then, in response to the problem of large state space and coexistence of discrete and continuous variables in the action space, the DDPG deep reinforcement learning algorithm is discretized and improved. Based on this, a multi-user computation offloading optimization method is proposed. Finally, this method is used to solve nonlinear programming problems. The simulation experimental results show that compared with existing algorithms, the proposed method can effectively reduce the probability of edge server overload and has good stability.

Key words: Mobile Edge Computing(MEC); Computation offloading; Deep reinforcement learning; Resource allocation

1 引言

通常，许多计算密集型、时延敏感型的计算需要消耗大量的计算资源，例如虚拟现实、自动驾驶和语音识别等。近年来这些应用不断往用户设备(User Equipment, UE)上迁移，但是由于UE存在计算资源和电池电量有限的先天不足，其中很多都无法在本地低延迟地处理这类计算任务，而需要远程服务器，甚至数据中心的支持才能完成[1,2]。移动边缘计算(Mobile Edge Computing, MEC)作为一种新型计算模式，通过将原本位于数据中心的计算资源下放到网络边缘，使UE上的计算密集型、时延敏感型应用能够将其计算任务卸载到移动边缘

收稿日期：2023-05-18；改回日期：2023-11-03；网络出版：2023-11-14
*通信作者： 李志华 zhli@jiangnan.edu.cn
基金项目：工业和信息化部智能制造项目(ZH-XZ-180004)，中央高校基本科研业务费专项资金(JUSRP211A41, JUSRP42003)
Foundation Items: The Ministry of Industry and Information Technology Manufacturing Project (ZH-XZ-180004), The Fundamental Research Funds for the Central Universities (JUSRP211A41, JUSRP42003)

服务器(MEC servers, MECs)上处理，从而实现更低的网络延迟，并起到显著节省核心网络的带宽、降低网络拥塞风险的作用[3]。但是，一方面，由于UE的相对位置和计算任务是随机的；另一方面，UE与MECs之间的无线通信链路具有不确定性。在这样的大背景下，研究UE高效的计算卸载策略具有一定的理论和技术双重挑战性。

文献[4–18]提出了多种计算卸载方案。可概括成以下3类：基于群体智能算法优化的计算卸载方案、基于博弈论的计算卸载方案，以及基于强化学习的计算卸载优化方案。文献[4–8]主要研究基于启发式算法的计算卸载方案，其主要目的是为了提高MECs的服务质量，然而，这些方法需要消耗大量的时间和计算资源来达到最终的近似最优解，并且初始参数偏多，设置也比较困难，不适用于实时性要求高的MEC场景；为了解决计算资源受限的时变环境中的计算卸载问题，文献[9–12]使用博弈理论设计了一系列计算卸载方法，这些方法能提高系统效用，并能有效地降低计算卸载的时延与能耗开销，但是，博弈理论通常涉及多个参与者之间的相互作用和关联决策，导致求解问题的复杂性较高，并且博弈理论是一种静态方法，通常在一段时间内进行决策，再根据结果进行调整，因此，博弈理论同样不适用于实时性要求高的MEC应用场景；为了增强计算卸载策略的实时性和自适应性，一些工作[13–18]致力于研究基于强化学习的计算卸载方案，实验结果表明，这些方法能在高实时性的动态应用环境中快速获得近似最优的计算卸载决策。

然而，文献[7–11,17,18]仅考虑单一MECs下的应用场景，限制了边缘网络的容量。相比之下，针对MECs密集部署场景，文献[4–6,12–16]对MECs负载不确定性关注不足，容易导致MECs出现过载现象。为此，需要在MECs密集部署场景下制定高效的多用户计算卸载和用户关联策略，以此避免MECs过载所导致的计算卸载时延和能耗显著增加。另外，由于强化学习对大规模、高维度和随机变化问题具有良好的自适应性，因此能够为MECs负载不确定性问题提供良好的解决方案。本文通过分析任务卸载、资源分配和用户关联策略对计算卸载过程中时延和能耗的综合影响，并考虑实际计算资源和计算费用等约束，把计算卸载问题建模成一个混合整数非线性规划(Mixed Integer NonLinear Programming, MINLP)问题。并对深度确定性策略梯度(Deep Deterministic Policy Gradient, DDPG)算法进行离散化改进，用离散化的算法求解上述优化问题。基于此，提出一种多用户计算卸载优化(Multi-user Computation Offloading Optimization, MCOO)方法。

本文主要贡献概况如下：(1)在MEC密集部署场景下，针对MECs负载的不确定性导致的MECs过载问题，提出一个多用户计算卸载优化模型；(2)根据时延和能耗的均衡优化建立效用函数，并以最大化卸载效用作为优化目标，将计算卸载问题转化为MINLP问题；(3)针对该问题的非凸性和关联策略的非连续性，对DDPG算法进行离散化改进，提出一种基于改进DDPG算法的MCOO方法。仿真实验结果表明，MCOO方法能快速获得计算卸载问题的近似最优解，并有效避免MECs过载现象发生，同时显著降低计算卸载的时延与能耗开销。与已有算法相比，MCOO方法具有一定的优势。

2 系统模型和问题描述

2.1 边缘网络模型

MEC边缘网络架构如图1所示。用集合$\mathcal{N}=\{U_1,U_2,\cdots,U_i,\cdots,U_N\}$表示用户层随机分布$N$个UE。边缘网络中存在$K$个与基站相连的MECs，用集合$\mathcal{M}=\{M_1,M_2,\cdots,M_j,\cdots,M_K\}$表示。整个通信周期$I$被划分为$T$个相等的时隙，用集合$\mathcal{T}=\{1,2,\cdots,t,\cdots,T\}$表示。在每个时隙开始时，$U_i$都会产生一个计算任务$\Omega_i^t$，其大小和所需要的计算资源是随机的。假设计算任务可以分割，即UE可以将计算任务一部分放在本地处理，另一部分卸载到MECs上处理。根据文献[19]，本文假设存在一个能够与所有MECs通信的控制代理。该控制代理可成功收集边缘网络的所有状态信息，例如每个MECs的资源占用情况，并根据资源占用等情况综合制定任务卸载决策，该决策主要包括任务卸载率、计算资源分配策略和关联策略。本文所用主要符号，如表1所示。

2.2 通信模型

在文献[20]的基础上，进一步考虑UE的相对位

图 1 边缘网络架构

表 1　变量符号及其含义

变量符号	含义	变量符号	含义
U_i	用户设备编号	P_i	U_i的传输功率
M_j	MEC服务器编号	σ^2	环境高斯白噪声
t	时隙编号	v_i	U_i的移动速度
f_i	U_i的CPU总频率	$x_{i,j}$	U_i的关联策略
φ	U_i的功率系数	λ_i^t	任务Ω_i^t的卸载率
D_i^t	任务大小	F_i^t	U_i分配到的计算资源
s	单位任务所需计算资源	$F_{\max}$	单个边缘服务器总频率
L_j	M_j的工作负载量		

置和不同通信链路间的干扰，建立如下通信模型。假设U_i以速度v_i随机移动，则在时隙t内，U_i的起始坐标为$\boldsymbol{p}_i^t=[x_i^t,y_i^t]$，$U_i$的结束坐标为$\boldsymbol{p}_i^{t+1}=[x_i^t+v_i\Delta_t\cos\gamma(t),y_i^t+v_i\Delta_t\sin\gamma(t)]$，其中$\Delta_t$是时隙长度，$\gamma(t)\in[0,2\pi]$是$U_i$的移动角度。$M_j$的位置固定，坐标为$\boldsymbol{q}_j=[x_j,y_j]$。$U_i$与基站之间的信道增益采用3GPP提供的标准化模型[21]，具体如式(1)所示

$$h_{i,j}^t=140.7+36.7\lg(d_{i,j}^t)+\mu \tag{1}$$

其中，μ表示影响信号接受功率的因素，服从对数正态分布$N(0,8\ \text{dB})$；$d_{i,j}^t$表示在时隙t时，U_i与M_j之间的距离，按式(2)计算

$$d_{i,j}^t=\|\boldsymbol{p}_i^t-\boldsymbol{q}_j\| \tag{2}$$

本文考虑了不同通信链路间的干扰，U_i与M_j之间的无线传输信道的信噪比按式(3)计算

$$\text{SNR}_{i,j}^t=\frac{P_ih_{i,j}^t}{\sigma^2+\sum\limits_{i\in\mathcal{N}\setminus\{i\}}P_ih_{i,j}^t} \tag{3}$$

其中，P_i表示U_i的发送功率，σ^2表示噪声功率。因此U_i与M_j之间的无线传输速率计算成如式(4)所示

$$R_{i,j}^t=B\log_2\left(1+\text{SNR}_{i,j}^t\right) \tag{4}$$

2.3　边缘网络中任务卸载时延和能耗的估算

通常任务部分卸载比完全本地执行和完全远程卸载更具有学术挑战性，所以本文采用部分卸载策略来处理U_i的计算任务。任务未卸载部分的处理时延估算成如式(5)所示

$$T_{\text{local},i}^t=\frac{(1-\lambda_i^t)D_i^ts}{f_i} \tag{5}$$

其中，$\lambda_i^t\in[0,1]$表示在时隙t时，U_i的任务卸载率；D_i^t表示计算任务输入数据的大小；s是单位任务所需要的计算资源；f_i是U_i自身的计算能力。未卸载部分的计算能耗估算成如式(6)所示

$$E_{\text{local},i}^t=\varphi f_i^3T_{\text{local},i}^t \tag{6}$$

其中，φ表示功率系数，取决于CPU芯片架构[22]。在MEC环境，服务器反馈的计算结果通常很小，计算反馈时间可忽略不计。因此，将任务部分卸载给MECs处理时，时延主要包括两个部分：一部分是传输时延，另一部分是任务处理时延，分别按式(7)、式(8)计算

$$T_{\text{tr},ij}^t=\frac{x_{i,j}\lambda_i^tD_i^t}{R_{i,j}^t} \tag{7}$$

$$T_{\text{mec},ij}^t=\frac{x_{i,j}\lambda_i^tD_i^ts}{F_i^t} \tag{8}$$

其中，$x_{i,j}\in\{0,1\}$表示U_i是否与M_j关联，F_i^t表示U_i分配到的计算资源。

考虑到任务部分卸载时任务的分割性，这样任务Ω_i^t完成的时延估算成如式(9)所示

$$T_{i,j}^t=\max\{T_{\text{local},i}^t,T_{\text{tr},ij}^t+T_{\text{mec},ij}^t\} \tag{9}$$

另外，考虑到MECs可能会出现过载的情况，任务Ω_i^t完成的总时延估算成如式(10)所示

$$T_{\text{total},ij}^t=\begin{cases}T_{i,j}^t, & \sum\limits_{i=1}^{N}\sum\limits_{j=1}^{K}x_{i,j}F_i^t\le F_{\max}\\ T_{i,j}^t+T_{\max}, & \text{其他}\end{cases} \tag{10}$$

其中，$F_{\max}$表示M_j总的计算资源，$T_{\max}$表示M_j过载时，U_i需要等待的最大时间。

将任务卸载到MECs上处理，能耗包括任务传输、任务处理两部分，分别按式(11)、式(12)估算

$$E_{\text{tr},ij}^t=P_iT_{\text{tr},ij}^t \tag{11}$$

$$E_{\text{mec},ij}^t=x_{i,j}\lambda_i^tD_i^ts\cdot\delta \tag{12}$$

其中，δ表示MECs单位CPU时钟周期能耗[23]。这样，任务Ω_i^t的总能耗按式(13)计算

$$E_{\text{total},ij}^t=E_{\text{local},i}^t+E_{\text{tr},ij}^t+E_{\text{mec},ij}^t \tag{13}$$

另外，服务商提供的MECs租赁费包括基本服务费和任务计算费用两部分。基本服务费取决于计算任务的大小，而任务计算费用取决于计算任务所占用的MECs资源。这样，任务Ω_i^t的成本费用如式(14)所示

$$O_i^t=\rho_{\text{d}}D_i^t\lambda_i^t+\rho_{\text{f}}F_i^t \tag{14}$$

其中，ρ_{d}表示单位任务量所需支付的费用，ρ_{f}表示单位计算资源所需支付的费用[24]。

2.4　问题定义

通常，在移动边缘计算应用场景中，时延决定用户体验，能耗直接影响服务商运营成本。受文

献[25]启发，本文通过对时延和能耗进行统合考虑，将任务本地处理和卸载到MECs处理之间的时延和能耗通过归一化后加权求和来建立效用函数。在此，时隙t时的卸载效用定义成如式(15)所示

$$\left.\begin{aligned} Q_t(\lambda_t,F_t,X_t)&=\beta_{\mathrm{t}}\frac{T_{\mathrm{l}}^t-T_{\mathrm{o}}^t}{T_{\mathrm{l}}^t}+\beta_{\mathrm{e}}\frac{E_{\mathrm{l}}^t-E_{\mathrm{o}}^t}{E_{\mathrm{l}}^t}\\ T_{\mathrm{o}}^t&=\sum_{i=1}^{N}\sum_{j=1}^{K}T_{\mathrm{total},ij}^t,E_{\mathrm{o}}^t=\sum_{i=1}^{N}\sum_{j=1}^{K}E_{\mathrm{total},ij}^t \end{aligned}\right\} \tag{15}$$

其中，$\lambda_t=\{\lambda_i^t|i\in\mathcal{N}\},F_t=\{F_i^t|\forall i\in\mathcal{N}\},X_t=\{x_{i,j}|\forall i\in\mathcal{N},\forall j\in\mathcal{M}\}$；$T_l^t$和$E_l^t$表示所有UE在本地执行计算任务的总时延和总能耗；$\beta_{\mathrm{t}},\beta_{\mathrm{e}}\in[0,1]$，并且$\beta_{\mathrm{t}}+\beta_{\mathrm{e}}=1$，分别表示时延、能耗对卸载效用影响的权重，可以根据UE的偏好进行设置。

进一步，在综合考虑MECs计算资源、服务商的计算成本等前提条件下，通过最大化卸载效用来优化任务卸载率、MECs资源的分配和用户关联策略。在此，把计算卸载问题定义成如式(16)所示的最大化系统卸载效用问题

$$\max_{\lambda,F,X}\sum_{t=1}^{T}Q_t=\sum_{t=1}^{T}\max_{\lambda_t,F_t,X_t}Q_t \tag{16}$$

$$\text{s.t. C1:}\lambda_i^t\in[0,1] \tag{16a}$$

$$\text{C2:}x_{i,j}\in\{0,1\} \tag{16b}$$

$$\text{C3:}T_{\mathrm{total},ij}^t\le\Delta_{\max} \tag{16c}$$

$$\text{C4:}O_i^t\le O_{\max} \tag{16d}$$

$$\text{C5:}\sum_{i=1}^{N}\sum_{j=1}^{K}x_{i,j}F_i^t\le F_{\max},\forall t\in\mathcal{T} \tag{16e}$$

显然，式(16)表示一个MINLP问题，即在C1～C5的约束下最大化卸载效用。在此，卸载效用的意义在于评估计算卸载和用户关联策略的优劣程度。其中，约束C1表示任务卸载率的取值范围；约束C2表示在时隙t，U_i与M_j是否关联；约束C3表示任务Ω_i^t所能容忍的最大时延；约束C4表示在时隙t时，任务的计算费用不能超过$O_{\max}$；约束C5表示与M_j关联的UE分配的计算资源之和不超过M_j总的计算资源。

3 基于DDPG的多用户计算卸载优化方法

为了求解式(16)中定义的MINLP问题。首先，结合实际MEC环境将式(16)描述为马尔可夫决策过程(Markov Decision Processes, MDP)；然后，设计一种状态归一化算法对状态进行预处理，并通过离散化动作空间对DDPG算法进行离散化改进；最后，综合状态归一化算法和改进后的DDPG算法提出了MCOO方法，用于求解式(16)所定义的优化问题。

3.1 计算卸载的马尔可夫决策过程

MDP是一种用于描述具有随机性质的序列决策问题的数学框架[26]，由一个4元组$(\mathcal{S},\mathcal{A},\mathcal{P},\mathcal{R})$组成，其中，$\mathcal{S}$是状态集合；$\mathcal{A}$是动作集合；$\mathcal{P}$表示在动作$\boldsymbol{a}_t\in\mathcal{A}$被执行之后，系统从当前状态$\boldsymbol{s}_t\in\mathcal{S}$转移到下一个状态$\boldsymbol{s}_{t+1}\in\mathcal{S}$的转移概率；$\mathcal{R}$是执行动作$a_t$的立即奖励函数。在本文中，把所有UE和MECs视为环境，控制代理扮演智能体角色。智能体通过与环境交互产生卸载决策。将UE的位置信息和产生的任务大小、MECs的负载量视为状态，把λ_t,F_t和X_t视为动作。智能体根据奖励函数来判断动作的优劣程度，并调整动作以优化卸载效用。因此，奖励函数需要与优化目标保持相同的变化趋势。基于以上考虑，本文定义了以下状态空间、动作空间和奖励函数。

(1)状态空间。智能体需要根据当前UE的位置信息和任务需求、MECs的资源状态选择动作。在时隙t时，系统状态s_t表示成如式(17)所示

$$\boldsymbol{s}_t=(\boldsymbol{p}_1^t,\cdots,\boldsymbol{p}_N^t,D_1^t,\cdots,D_N^t,L_1^t,\cdots,L_M^t) \tag{17}$$

其中，$\boldsymbol{p}_i^t$表示U_i的位置信息，D_i^t表示U_i随机生成的任务大小，L_j^t表示M_j的负载量。它们的数量级存在较大差异，这也是在设计算法时需要考虑的因素之一。

(2)动作空间。智能体在时隙t时根据当前系统状态和观察到的环境来选择动作，包括任务卸载率、分配的计算资源和关联策略，动作$\boldsymbol{a}_t$如式(18)所示

$$\boldsymbol{a}_t=(\lambda_1^t,\cdots,\lambda_N^t,F_1^t,\cdots,F_N^t,x_1^t,\cdots,x_N^t) \tag{18}$$

其中，由于任务卸载率、分配的计算资源是连续型变量，而关联策略是离散型变量，因此智能体选择的动作连续型和离散型并存，如何解决此类问题具有一定挑战性。

(3)奖励函数。在强化学习中，智能体的目标就是在给定环境下选择最优的动作以最大化奖励，而本文的优化目标是最大化计算卸载效用，因此奖励函数的值与卸载效用的大小呈正相关。在时隙t时，奖励函数定义成如式(19)所示

$$r_t=\beta_{\mathrm{t}}\frac{T_{\mathrm{l}}^t-T_{\mathrm{o}}^t}{T_{\mathrm{l}}^t}+\beta_{\mathrm{e}}\frac{E_{\mathrm{l}}^t-E_{\mathrm{o}}^t}{E_{\mathrm{l}}^t}+\mathrm{punish} \tag{19}$$

其中，punish是惩罚函数，若系统中的变量违反约束C1～C5，则将受到相应的惩罚。

3.2 离散化DDPG算法

由于状态空间$\boldsymbol{s}_t$中变量的量纲不同，动作空间

a_t中连续型和离散型变量共存，而现有强化学习算法解决此类问题具有一定挑战性。DDPG算法通过将Actor-Critic框架和深度学习技术相结合，在解决连续动作控制问题中具有较好表现[27]。鉴于此，本节设计一种归一化算法对状态进行预处理，并对DDPG算法进行离散化改进，以便快速求解式(16)定义的优化问题。

在DDPG算法中，Actor网络用于输出当前状态s_t下采取的动作$\boldsymbol{a}_t$，而Critic网络则用于评估动作a_t的优劣。通常，DDPG算法使用两个不同的深度神经网络分别逼近Actor网络$\mu(\boldsymbol{s}_t|\theta_\mu)$和Critic网络$Q(\boldsymbol{s}_t,\boldsymbol{a}_t|\theta_Q)$；此外，Actor网络和Critic网络都包含与它们结构相同但参数不同的目标网络：Actor目标网络$\mu'(\boldsymbol{s}_t|\theta'_\mu)$和Critic目标网络$Q'(\boldsymbol{s}_t,\boldsymbol{a}_t|\theta'_Q)$。在DDPG算法中，通常还会有一个大小固定的缓冲区B_m，用于存储智能体与环境交互所得到的经验元组。在每次迭代过程中，Critic网络$Q(\boldsymbol{s}_t,\boldsymbol{a}_t|\theta_Q)$通过最小化损失函数来更新参数$\theta_Q$，损失函数如式(20)所示

$$L(\theta_Q)=\mathbb{E}\left[(y_t-Q(\boldsymbol{s}_t,\boldsymbol{a}_t|\theta_Q))^2\right] \tag{20}$$

其中，y_t为目标值，按式(21)计算

$$y_t=r_t+\gamma Q'(\boldsymbol{s}_{t+1},\mu'(\boldsymbol{s}_{t+1}|\theta'_\mu)|\theta'_Q) \tag{21}$$

在初始化和相关步骤完成之后，Actor网络会输出根据状态s_t直接生成的动作$\boldsymbol{a}_t=\mu(\boldsymbol{s}_t)$。执行动作$\boldsymbol{a}_t$，首先，智能体可以观测到下一个状态$\boldsymbol{s}_{t+1}$和即时奖励$r_t$；然后，将经验元组$(\boldsymbol{s}_t,\boldsymbol{a}_t,r_t,\boldsymbol{s}_{t+1})$保存在$B_m$中；最后，智能体会从$B_m$中随机采样一批固定数量的经验元组$(\boldsymbol{s}_k,\boldsymbol{a}_k,r_k,\boldsymbol{s}_{k+1})$，并将这批经验元组分别输入到主网络和目标网络中。$\mu(\boldsymbol{s}_t|\theta_\mu)$使用策略梯度来更新参数$\theta_\mu$，更新策略具体如式(22)所示

$$\nabla_{\theta_\mu}J\approx\mathbb{E}\left[\nabla_{\boldsymbol{a}}Q(\boldsymbol{s},a|\theta_Q)|_{\boldsymbol{s}=\boldsymbol{s}_j,a=\mu(s_j)}\nabla_{\theta_\mu}\mu(\boldsymbol{s}|\theta_\mu)|_{\boldsymbol{s}=\boldsymbol{s}_j}\right] \tag{22}$$

在迭代过程中，DDPG算法使用一个常量$\tau\in[0,1]$来更新目标网络的参数，θ'_μ和θ'_Q分别按式(23)、式(24)更新

$$\theta'_\mu=\tau\theta_\mu+(1-\tau)\theta'_\mu \tag{23}$$

$$\theta'_Q=\tau\theta_Q+(1-\tau)\theta'_Q \tag{24}$$

在状态$\boldsymbol{s}_t$中，位置信息和任务大小变量位于不同的范围内，这会导致算法训练不收敛。为解决此问题，本文设计一种状态归一化 (State Normalization, SN) 算法来对观测状态进行预处理，从而更高效地训练深度神经网络。SN算法的伪代码描述如算法1所示。具体而言，SN算法以每个变量的最大值的倒数作为比例因子，对观测到的状态进行归一化处理，这样有利于将每个时隙t内U_i的任务大小、x坐标和y坐标等变量缩放到相同的尺度上。

在原始动作$\boldsymbol{a}_t=\mu(\boldsymbol{s}_t)$中，由于动作值精度不够高，容易导致奖励值$r_t$震荡。为了解决这个问题，可以使用式(25)添加高斯噪声n_t来重构动作a_t

$$\boldsymbol{a}_t=\mathrm{clip}(\mu(\boldsymbol{s}_t)+n_t,a_{\mathrm{low}},a_{\mathrm{high}}) \tag{25}$$

其中，n_t服从高斯分布$n_t\sim\mathbb{N}(\mu_t,\sigma_t^2)$，$\mu_t$是均值，$\sigma_t^2$是标准差；clip函数用于限制动作值在动作空间的范围内；a_{low}和a_{high}分别表示动作空间的最小值和最大值。在动作$\boldsymbol{a}_t$中，关联策略是离散变量。然而，传统的DDPG算法只能用于优化连续型变量的取值，无法直接应用于离散动作空间。因此，本文采用动作编码(Action Encoding, AE)的方式对传统的DDPG算法进行离散化改进。动作编码的基本思想是将连续动作空间划分为多个离散动作区间，并为每个区间分配一个唯一的编码。这个编码通常是一个整数，可用作代表该动作的离散状态的值。

本文主要使用以下步骤将连续动作空间进行离散化：首先，将连续动作空间划分为K个离散动作区间，每个区间的大小相等，对应UE与K个MEC服务器的关联策略；然后，为每个区间分配一个唯一的整数编码，这些编码将用于代表动作的离散状态值。具体而言，针对每个连续动作，使用式(26)将其映射到所属区间的编码

$$\mathrm{encode}(a)=\left\lfloor\frac{a-a_{\min}}{\Delta_a}\right\rfloor \tag{26}$$

其中，$\mathrm{encode}(a)$表示动作a对应的离散编码，$\lfloor x\rfloor$是向下取整函数，$a_{\min}$是动作空间的最小值，Δ_a表示每个区间的大小；最后，使用动作编码代替连续动作空间中的动作，并将其传递给DDPG算

算法1　状态归一化算法

输入: Unnormalized variables: $\boldsymbol{s}_t=(\boldsymbol{p}_1^t,\cdots,\boldsymbol{p}_N^t,D_1^t,\cdots,D_N^t,L_1^t,\cdots,L_M^t)$, Scale factors: $\rho=(\rho_x,\rho_y,\rho_w,\rho_l)$

输出: Normalized variables: $(\boldsymbol{p'}_1^t,\cdots,\boldsymbol{p'}_N^t,D'_1{}^t,\cdots,D'_N{}^t,L'_1{}^t,\cdots,L'_M{}^t)$

(1) $x'^t_i=x_i^t*\rho_x,\forall i,\ y'^t_i=y_i^t*\rho_y,\forall i,\ D'^t_i=D_i^t*\rho_w,\ \forall i,\ L'^t_j=L_j^t*\rho_l,\ \forall j$ //*对状态进行归一化处理

(2) return $\hat{s}_t=(\boldsymbol{p'}_1^t,\cdots,\boldsymbol{p'}_N^t,D'_1{}^t,\cdots,D'_N{}^t,L'_1{}^t,\cdots,L'_M{}^t)$

法中的目标网络进行训练。动作编码的伪代码描述如算法2所示。

算法2的计算开销主要来自于第4～7行，即对动作编码过程。因此，AE算法的时间复杂度为$O(N)$。

综上所述，本文对经典的DDPG算法进行离散化改进，基于此进一步提出一种MCOO方法，用于求解式(16)定义的优化问题。MMCO方法首先使用参数θ_μ和θ_Q分别初始化Actor深度神经网络$\mu(\boldsymbol{s}_t|\theta_\mu)$和Critic深度神经网络$Q(\boldsymbol{s}_t,\boldsymbol{a}_t|\theta_Q)$，并初始化目标网络和经验重放缓冲区；然后，调用算法1对系统状态进行归一化处理，使用式(25)获得动作a_t，并调用算法2对连续动作空间离散化，从经验重放缓冲区中随机抽取小批量经验元组，用于更新主网络参数和目标网络参数；最后，返回每次迭代过程中的卸载决策和卸载效用。MCOO方法的伪代码描述如算法3所示。

MCOO方法的计算开销主要来自于第2～21行，即迭代过程和在第7行调用算法2产生离散化动作所消耗的时间，因此，MCOO方法的时间复杂度为$O(LTN)$。

算法2 动作编码算法

输入: a //*连续动作

输出: a_{dis} //*对应的离散编码

(1) $a_{\text{num}}=K$

(2) $a_{\min}=0, a_{\max}=1$

(3) $\Delta_a=(a_{\max}-a_{\min})/(a_{\text{num}}-1)$

(4) for each a do

(5) $a'=\left\lfloor\dfrac{a-a_{\min}}{\Delta_a}\right\rfloor$ //*动作编码

(6) $a_{\text{dis}}=\max(0,\min(a_{\text{num}}-1,a'))$

(7) end for

(8) return a_{dis}

4 实验结果与分析

本节主要通过实验来验证MCOO方法的有效性和高效性。首先介绍评价指标和仿真参数的设

算法3 多用户计算卸载优化方法

输入: Actor learning rate α_{Actor}, critic learning rate α_{Critic}, Soft update factor τ.

输出：a,Q //*卸载决策(任务卸载率、分配的计算资源和关联策略)，卸载效用

(1) $\mu(s_t|\theta_\mu)\leftarrow\theta_\mu$ and $Q(\boldsymbol{s}_t,\boldsymbol{a}_t|\theta_Q)\leftarrow\theta_Q$, $\theta'_\mu\leftarrow\theta_\mu$ and $\theta'_Q\leftarrow\theta_Q$ //*初始化主网络和目标网络

Initialize the experience replay buffer B_m //*初始化经验重放缓冲区暂存经验元组

(2) for episode=1 to Ldo

(3) Initialize system environment

(4) for slot=1 to Tdo

(5) $\hat{\boldsymbol{s}}_t\leftarrow\text{SN}(\boldsymbol{s}_t,\rho)$ //*调用算法1对状态$\boldsymbol{s}_t$预处理

(6) Get the action from equation 式(25)

(7) $\boldsymbol{a}'_t\leftarrow\text{AE}(\boldsymbol{a}_t)$ //*调用算法2离散化动作

(8) perform action $\boldsymbol{a}'_t$and observer next state $\boldsymbol{s}_{t+1}$, Get reward with equation 式(19)

(9) $\hat{\boldsymbol{s}}_{t+1}\leftarrow\text{SN}(\boldsymbol{s}_{t+1},\rho)$ //*调用算法1对状态$\boldsymbol{s}_{t+1}$预处理

(10) if $B_m$is not full then

(11) Store transition $(\boldsymbol{s}_t,\boldsymbol{a}_t,r_t,\hat{\boldsymbol{s}}_{t+1})$ in replay buffer B_m

(12) else

(13) Randomly sample a mini-batch from B_m

(14) Calculate target value y_t with equation 式(21)

(15) Use equation 式(20) to minimize the loss and update the θ_Q

(16) Update the θ_μ by the sampled policy gradient with equation 式(22)

(17) Soft update the θ'_μ and θ'_Q according to equation 式(23) and 式(24)

(18) end if

(19) end for

(20) Use equation 式(15) to get offloading utility Q

(21) end for

(22) return a,Q

置。然后在不同场景下验证所提方法的性能，并与其他算法进行比较。

4.1 评价指标

为了评估MCOO方法的性能，本文采用平均卸载效用、累积奖励、MECs过载概率和平均响应率作为评价指标。

使用平均卸载效用来评价在算法的迭代过程中所有UE的卸载效用，具体按式(27)计算

$$q=\begin{cases}\dfrac{\sum\limits_{t=1}^{T}Q_t}{T}, Q>0\\ 0, \quad \text{其他}\end{cases} \tag{27}$$

累积奖励是一次迭代过程中所有时隙的立即奖励的累加和，主要用于评估算法在迭代过程中是否找到可行解，具体按式(28)计算

$$R=\sum_{t=1}^{T}r_t \tag{28}$$

受文献[28]启发，本文使用式(29)来计算任务Ω_i^t的响应率，用以衡量任务执行的效率和及时性，

$$\mathrm{RES}_{\Omega_i^t}=\begin{cases}1, & T^t_{\mathrm{total},ij}\le T_g\\ 1-\dfrac{1}{1+\mathrm{e}^{\alpha\Delta_{t1}}}, & T_g<T^t_{\mathrm{total},ij}\le T_{\mathrm{avg}}\\ \dfrac{1}{1+\mathrm{e}^{\alpha\Delta_{t2}}}, & T_{\mathrm{avg}}<T^t_{\mathrm{total},ij}\le \Delta_{\max}\\ 0, & T^t_{\mathrm{total},ij}>\Delta_{\max}\end{cases}$$

$$T_{\mathrm{avg}}=\frac{T_{\mathrm{g}}+\Delta_{\max}}{2}$$

$$\Delta_{\mathrm{t}1}=(T_{\mathrm{avg}}-T^t_{\mathrm{total},ij})/(T_{\mathrm{avg}}-T_{\mathrm{g}})$$

$$\Delta_{\mathrm{t}2}=(T^t_{\mathrm{total},ij}-T_{\mathrm{avg}})/(\Delta_{\max}-T_{\mathrm{avg}}) \tag{29}$$

其中，T_{g}是理想完成时延，$\Delta_{\max}$是UE所能忍受的最大时延，T_{avg}是T_{g}和$\Delta_{\max}$的平均值。

过载概率通常是指在特定的资源负载下，MECs的CPU使用率、内存使用率等超过其正常范围的概率。当MECs过载时，会出现性能下降、响应时间延长等问题，从而导致计算卸载过程中时延和能耗显著增加。本文主要关注MECs的CPU使用率，因此，M_j的过载概率可以估算成如式(30)所示

$$P(L_j>F_{\max})=1-\Theta(F_{\max}) \tag{30}$$

其中，L_j是M_j的负载变量，$F_{\max}$表示M_j总的计算资源，$\Theta(F_{\max})$是$L_j\le F_{\max}$的概率。

4.2 实验设置

实验使用 Python 3.7 和 TensorFlow 1.14.0 作为实验环境，并在 Windows 11 操作系统下运行，处理器为Intel Core i7-9 700 CPU@3.00 GHz。考虑MECs密集部署场景下，在基站覆盖范围内随机分布N个用户设备。在每个时隙，用户设备的移动速度在区间[0,5]内随机取值。系统总带宽B为40 MHz，环境高斯白噪声功率σ^2为–100 dBm，U_i的发射功率P_i为0.1 W，单位任务所需要的CPU周期s为500 cycle/bit。U_i和M_j的计算能力分别设置为0.5 GHz和10 GHz。表2列出了实验中用到的主要参数。为有效消除偶然误差，提高实验结果的可信度，实验结果数据均取10次实验的平均值。

4.3 MCOO方法的有效性

本节主要使用平均卸载效用、累积奖励、MECs过载概率和平均响应率4个评价指标来验证MCOO方法的有效性。在此之前，为了分析不同的β_{t}和β_{e}对MCOO方法实验结果的影响，如表3所示，本文设计了9组不同的β_{t}和β_{e}值来比较MCOO方法产生的时延、能耗和平均卸载效用。可以看出随着时延权重的增加，平均卸载效用也在增加。为确保时延和能耗都能够维持较低水平，本文将时延与能耗设置相同的权重，即$\beta_{\mathrm{t}}=\beta_{\mathrm{e}}=0.5$。

图2显示了没有对状态进行归一化处理或离散化动作空间时对MCOO方法性能的影响，其中

表 2　仿真参数设置

符号	值	定义
B	40	信道总带宽(MHz)
N	{5,10,20,30,40}	用户设备数
K	5	MEC服务器个数
v_i	[0,5]	U_i的移动速度(m/s)
P_i	100	U_i的传输功率(mW)
σ^2	–100	环境高斯白噪声(dBm)
f_i	0.5	U_i的CPU总频率(GHz)
$F_{\max}$	10	单个MEC服务器总频率(GHz)
φ	10^{-26}	功率系数
D_i^t	(1.5,2)	任务Ω_i^t的大小(Mbit)
s	500	所需计算资源(cycle/bit)

表 3　不同β_{t}和β_{e}下的实验结果

β_{t}和β_{e}的取值	时延(s)	能耗(J)	平均卸载效用
$\beta_{\mathrm{t}}=0.1,\beta_{\mathrm{e}}=0.9$	369	394	0.528
$\beta_{\mathrm{t}}=0.2,\beta_{\mathrm{e}}=0.8$	343	412	0.534
$\beta_{\mathrm{t}}=0.3,\beta_{\mathrm{e}}=0.7$	337	431	0.544
$\beta_{\mathrm{t}}=0.4,\beta_{\mathrm{e}}=0.6$	321	441	0.549
$\beta_{\mathrm{t}}=0.5,\beta_{\mathrm{e}}=0.5$	308	456	0.554
$\beta_{\mathrm{t}}=0.6,\beta_{\mathrm{e}}=0.4$	293	464	0.575
$\beta_{\mathrm{t}}=0.7,\beta_{\mathrm{e}}=0.3$	277	479	0.578
$\beta_{\mathrm{t}}=0.8,\beta_{\mathrm{e}}=0.2$	256	488	0.613
$\beta_{\mathrm{t}}=0.9,\beta_{\mathrm{e}}=0.1$	245	511	0.628

w.o.SN方法表示未对状态进行归一化处理，Random方法表示未对DDPG算法进行离散化改进，采用随机关联策略。从图2(a)、图2(b)和图2(d)可以看出，MCOO方法在重放内存B_m满了之后，即迭代次数大于250后，平均卸载效用、累积奖励和平均响应率呈快速增长趋势，当迭代次数超过600时逐渐趋于稳定。如果在训练策略之前没有对状态进行归一化处理，由于不同变量之间存在较大的量级差异，会导致梯度在更新时来回震荡，导致MCOO方法性能降低。如图2(c)所示，由于Random方法在求解出卸载率和分配的计算资源后，采用随机关联策略，使得MECs过载概率相对较高。

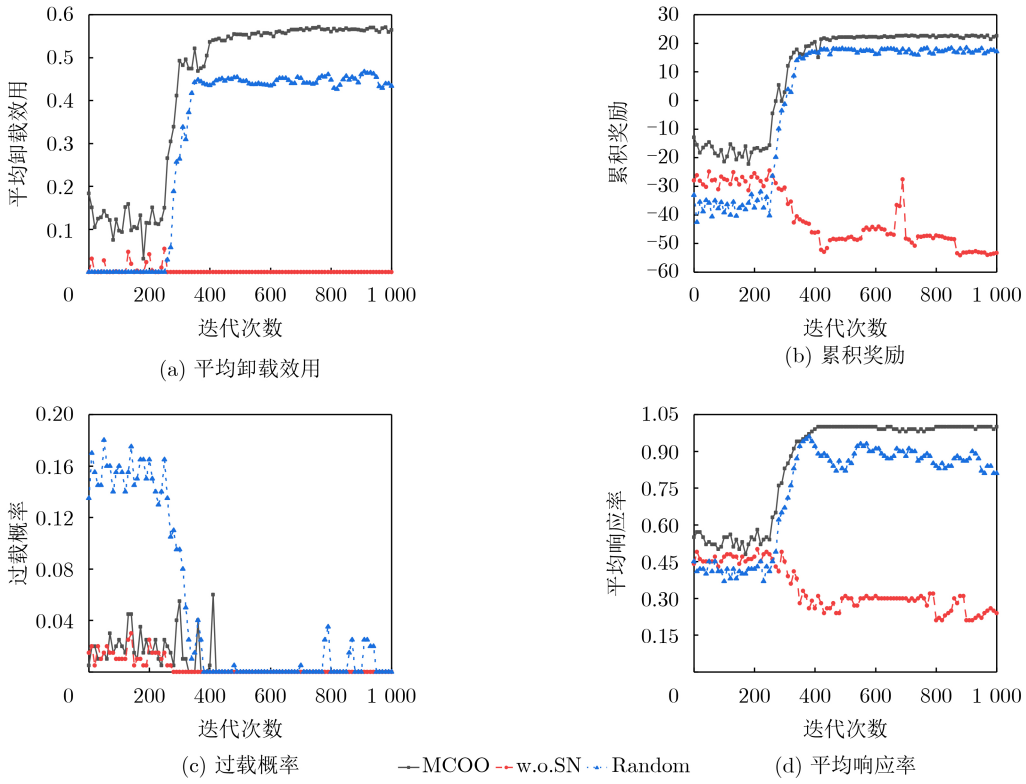
从图2可知，当MECs资源充足时，相较于Random方法，MCOO方法在平均卸载效用和平均响应率方面分别提高了约26%和14%，同时有效降低了MECs的过载概率。这些实验结果表明MCOO方法具有良好的收敛性和相对较强的实用性。

4.4 MCOO方法的高效性

在本节中主要通过设计实验来验证MCOO方法的高效性。首先，在s和UE数量固定的情况下，将MCOO与以下几种算法比较：AC[16]、始终不卸载(No Offloading Scheme, NOS)、完全卸载(Offloading Completely Scheme, OCS)、DQN[18]和Random。在NOS中，所有UE选择在本地执行计算任务。在OCS中，UE将计算任务全部卸载到MECs，计算资源按随机策略分配，此外，AC和OCS算法采用与Random方法相同的随机关联策略。然后，分析了不同算法随s变化时的性能表现。最后，还分析了不同算法随UE数量变化时的性能特点。

图3显示了每个算法随迭代次数变化时的性能表现。在图3(b)中，NOS策略累积奖励最低，这是因为其违反了约束C3。另外，由于NOS算法所有计算任务都在本地执行，因此其平均卸载效用和MECs过载概率始终为零。在整个迭代过程中，AC算法无法找到适当的卸载决策和资源分配策略，导致其性能较低。而DQN算法由于找不到合理的用户关联策略，使得部分MECs过载，导致其性能偏低。如图3(a)、图3(b)所示，MCOO方法和OCS算法在平均卸载效用和累积奖励方面表现相似。这是因为在带宽和计算资源充足的情况下，将任务全部卸载到MECs上处理可以降低时延和能耗。然而，在图3(c)、3(d)中，MCOO方法获得了最低的过载概率和最高的平均响应率。综合来看，当MECs资源充足时，相较于AC, OCS和DQN算法，MCOO方法在平均卸载效用方面分别提高了约59%, 5%和56%；在平均响应率方面分别提高了约55%, 14%和28%。同时有效降低了MECs的过载概率。

当任务大小不变，s减小时。如图4(b)、图4(c)所示，完成任务所需的时延和能耗自然而然地会大大降低。如图4(a)所示，与时延和能耗相关的平均



(a) 平均卸载效用 (b) 累积奖励 (c) 过载概率 (d) 平均响应率

图 2 没有对状态进行归一化处理或离散化动作空间时的性能对比

卸载效用相应地也会大幅降低。由于完成任务所需时延的降低，平均响应率逐渐升高，并达到最大值，如图4(d)所示。综合来看，所提出的MCOO方法能够保持较低的时延和能耗，同时保证相对较高的平均卸载效用和平均响应率。

图5、图6展示了不同UE数量下各种算法的时

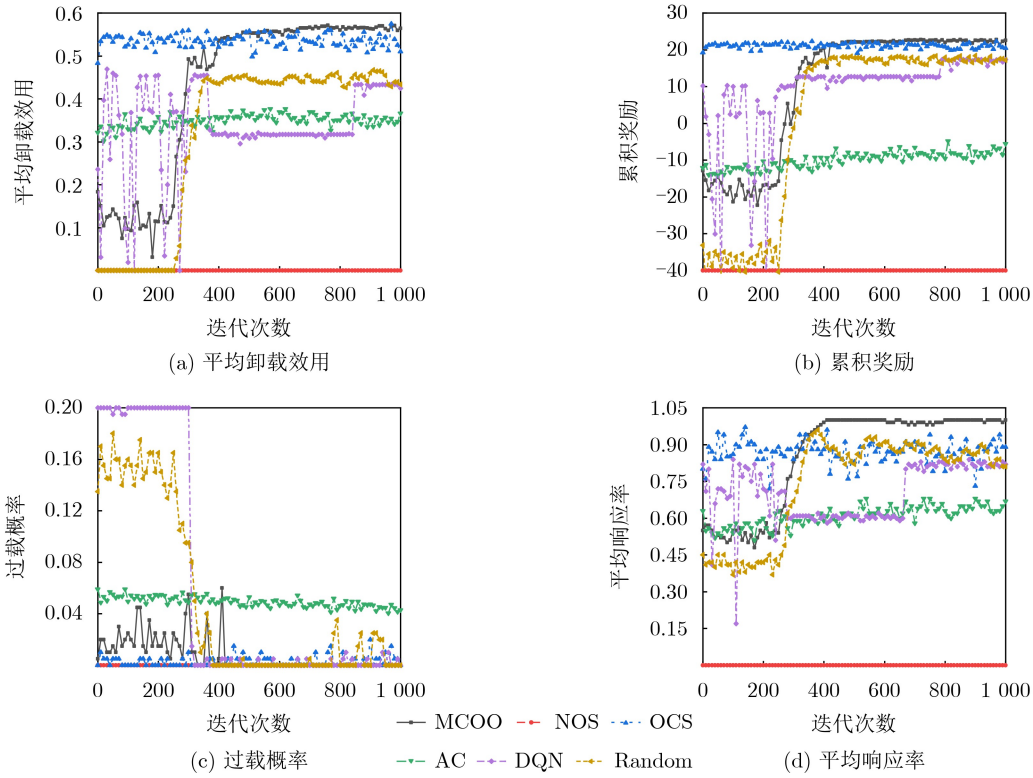


图 3 不同算法随迭代次数变化时的性能对比

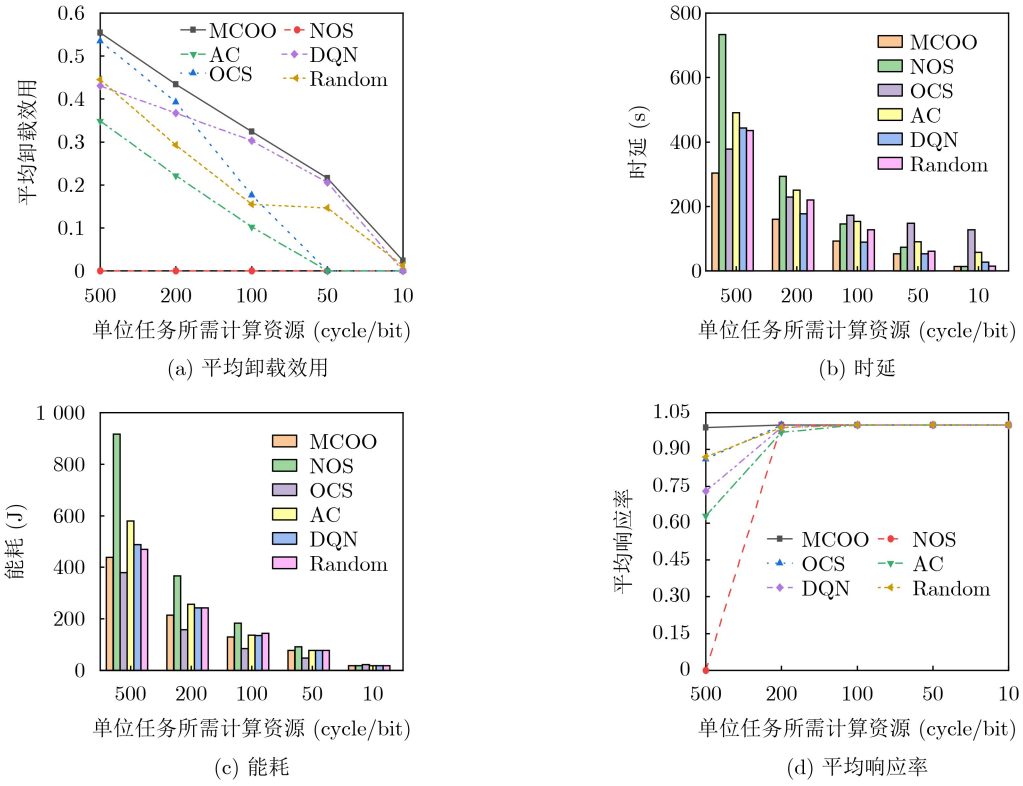


图 4 不同算法随s变化时的性能对比

延和能耗情况。当UE数量较少时，MECs的带宽和计算资源相对充足。因此，相较于NOS算法，其他算法产生的时延与能耗相对较低。然而，当UE数量超过30时，MECs的带宽和计算资源不足。大量任务被卸载到MECs上，导致部分MECs过载，进而引发显著的时延和能耗增加。总体而言，提出的MCOO方法在平衡时延和能耗方面表现出色。随着UE数量的增加，MCOO方法始终保持较低的时延和能耗水平。

如图7所示，随着UE数量的增加，每个算法的平均卸载效用和平均响应率都在逐渐下降，而过载概率逐渐上升。这是因为MECs带宽和计算资源有限，随着UE数量的增加会使得部分MECs过载，导致计算卸载过程中时延和能耗增加。MCOO方法能够根据MECs的资源占用情况制定合适的卸载决策和资源分配策略，同时确定合理的用户关联策略。因此，随着UE数量的增加MCOO方法仍然能够获得较高的平均卸载效用和较低的过载概率。综合来看，在UE数量达到最大时，AC和DQN算法表现欠佳，相较于UE数量为30时，MCOO方法、OCS算法和Random方法在平均卸载效用方面分别降低了约20%, 73%和75%；在平均响应率方面分别降低了约3%, 82%和35%。此外，MCOO方法的过载概率始终较低。

通过上述实验结果分析可以发现，与其他算法相比，MCOO方法能有效提高平均卸载效用和平均响应率，同时降低MECs的过载概率。因此，将MCOO方法用于计算卸载和用户关联策略的制定能够保障MEC系统的高效运行。

5 结束语

本文针对MECs密集部署场景研究并提出一种多用户计算卸载优化模型。本模型旨在解决现有研究工作中对MECs负载不确定性关注不足导致其过载的问题。为了高效求解优化模型，本文结合动作编码对经典的DDPG强化学习算法进行离散化改进，基于此提出一种多用户计算卸载优化方法。该方法通过优化任务卸载率、计算资源分配和关联策略3个决策变量，以最大化任务卸载效用，并考虑MECs计算资源和UE最大可容忍时延等约束。实验结果表明，本文所提出方法能在时变的MEC环境中快速得到近似最优解，达到较高的平均卸载效用和平均响应率，同时保持较低的过载概率，与已有算法相比，具有更高的效率和精度。然而，本文方法也存在一些不足之处。未来将进一步结合服务缓存和UE的偏好，制定更加有效的用户关联策略。

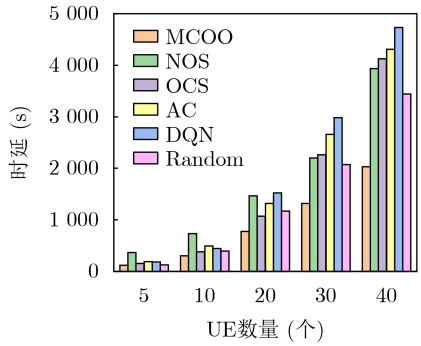


图 5 不同UE数量下的时延对比图

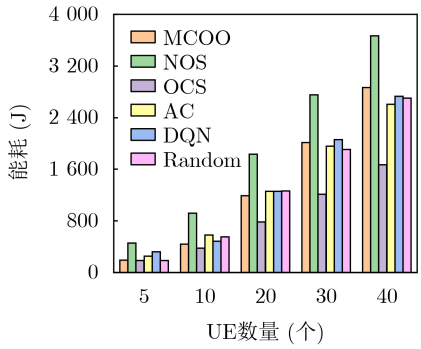


图 6 不同UE数量下的能耗对比图

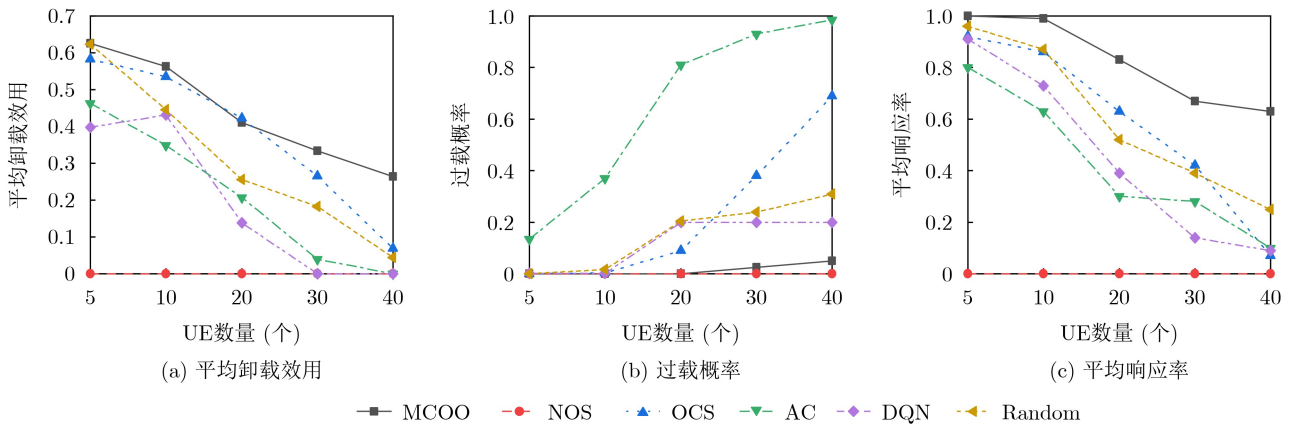


图 7 不同算法随UE数量变化时的性能对比

参 考 文 献

[1] ZHOU Zhi, CHEN Xu, LI En, *et al*. Edge intelligence: Paving the last mile of artificial intelligence with edge computing[J]. *Proceedings of the IEEE*, 2019, 107(8): 1738–1762. doi: 10.1109/JPROC.2019.2918951.

[2] GERARDS M E T, HURINK J L, and KUPER J. On the interplay between global DVFS and scheduling tasks with precedence constraints[J]. *IEEE Transactions on Computers*, 2015, 64(6): 1742–1754. doi: 10.1109/TC.2014.2345410.

[3] SADATDIYNOV K, CUI Laizhong, ZHANG Lei, *et al*. A review of optimization methods for computation offloading in edge computing networks[J]. *Digital Communications and Networks*, 2023, 9(2): 450–461. doi: 10.1016/j.dcan.2022.03.003.

[4] SUN Jiannan, GU Qing, ZHENG Tao, *et al*. Joint optimization of computation offloading and task scheduling in vehicular edge computing networks[J]. *IEEE Access*, 2020, 8: 10466–10477. doi: 10.1109/ACCESS.2020.2965620.

[5] LIU Hui, NIU Zhaocheng, DU Junzhao, *et al*. Genetic algorithm for delay efficient computation offloading in dispersed computing[J]. *Ad Hoc Networks*, 2023, 142: 103109. doi: 10.1016/j.adhoc.2023.103109.

[6] ALAMEDDINE H A, SHARAFEDDINE S, SEBBAH S, *et al*. Dynamic task offloading and scheduling for low-latency IoT services in multi-access edge computing[J]. *IEEE Journal on Selected Areas in Communications*, 2019, 37(3): 668–682. doi: 10.1109/JSAC.2019.2894306.

[7] BI Suzhi, HUANG Liang, and ZHANG Y J A. Joint optimization of service caching placement and computation offloading in mobile edge computing systems[J]. *IEEE Transactions on Wireless Communications*, 2020, 19(7): 4947–4963. doi: 10.1109/TWC.2020.2988386.

[8] YI Changyan, CAI Jun, and SU Zhou. A multi-user mobile computation offloading and transmission scheduling mechanism for delay-sensitive applications[J]. *IEEE Transactions on Mobile Computing*, 2020, 19(1): 29–43. doi: 10.1109/TMC.2019.2891736.

[9] MITSIS G, TSIROPOULOU E E, and PAPAVASSILIOU S. Price and risk awareness for data offloading decision-making in edge computing systems[J]. *IEEE Systems Journal*, 2022, 16(4): 6546–6557. doi: 10.1109/JSYST.2022.3188997.

[10] ZHANG Kaiyuan, GUI Xiaolin, REN Dewang, *et al*. Optimal pricing-based computation offloading and resource allocation for blockchain-enabled beyond 5G networks[J]. *Computer Networks*, 2022, 203: 108674. doi: 10.1016/j.comnet.2021.108674.

[11] TONG Zhao, DENG Xin, MEI Jing, *et al*. Stackelberg game-based task offloading and pricing with computing capacity constraint in mobile edge computing[J]. *Journal of Systems Architecture*, 2023, 137: 102847. doi: 10.1016/j.sysarc.2023.102847.

[12] 张祥俊, 伍卫国, 张弛, 等. 面向移动边缘计算网络的高能效计算卸载算法[J]. 软件学报, 2023, 34(2): 849–867. doi: 10.13328/j.cnki.jos.006417.
ZHANG Xiangjun, WU Weiguo, ZHANG Chi, *et al*. Energy-efficient computing offloading algorithm for mobile edge computing network[J]. *Journal of Software*, 2023, 34(2): 849–867. doi: 10.13328/j.cnki.jos.006417.

[13] YAO Liang, XU Xiaolong, BILAL M, *et al*. Dynamic edge computation offloading for internet of vehicles with deep reinforcement learning[J]. *IEEE Transactions on Intelligent Transportation Systems*, 2023, 24(11): 12991–12999. doi: 10.1109/TITS.2022.3178759.

[14] SADIKI A, BENTAHAR J, DSSOULI R, *et al*. Deep reinforcement learning for the computation offloading in MIMO-based Edge Computing[J]. *Ad Hoc Networks*, 2023, 141: 103080. doi: 10.1016/j.adhoc.2022.103080.

[15] TANG Ming and WONG V W S. Deep reinforcement learning for task offloading in mobile edge computing systems[J]. *IEEE Transactions on Mobile Computing*, 2022, 21(6): 1985–1997. doi: 10.1109/TMC.2020.3036871.

[16] CHENG Nan, LYU Feng, QUAN Wei, *et al*. Space/aerial-assisted computing offloading for IoT applications: A learning-based approach[J]. *IEEE Journal on Selected Areas in Communications*, 2019, 37(5): 1117–1129. doi: 10.1109/JSAC.2019.2906789.

[17] ZHOU Huan, JIANG Kai, LIU Xuxun, *et al*. Deep reinforcement learning for energy-efficient computation offloading in mobile-edge computing[J]. *IEEE Internet of Things Journal*, 2022, 9(2): 1517–1530. doi: 10.1109/JIOT.2021.3091142.

[18] WANG Yunpeng, FANG Weiwei, DING Yi, *et al*. Computation offloading optimization for UAV-assisted mobile edge computing: A deep deterministic policy gradient approach[J]. *Wireless Networks*, 2021, 27(4): 2991–3006. doi: 10.1007/s11276-021-02632-z.

[19] ALE L, ZHANG Ning, FANG Xiaojie, *et al*. Delay-aware and energy-efficient computation offloading in mobile-edge computing using deep reinforcement learning[J]. *IEEE Transactions on Cognitive Communications and Networking*, 2021, 7(3): 881–892. doi: 10.1109/TCCN.2021.3066619.

[20] DAI Yueyue, XU Du, ZHANG Ke, *et al*. Deep reinforcement learning for edge computing and resource allocation in 5G beyond[C]. The IEEE 19th International Conference on Communication Technology, Xian, China,

2019: 866–870. doi: 10.1109/ICCT46805.2019.8947146.

[21] 3GPP. TR 36.814 v9.0. 0. Further advancements for E-UTRA physical layer aspects[S]. 2010.

[22] WANG Yanting, SHENG Min, WANG Xijun, *et al*. Mobile-edge computing: Partial computation offloading using dynamic voltage scaling[J]. *IEEE Transactions on Communications*, 2016, 64(10): 4268–4282. doi: 10.1109/TCOMM.2016.2599530.

[23] ZHANG Ke, MAO Yuming, LENG Supeng, *et al*. Energy-efficient offloading for mobile edge computing in 5G heterogeneous networks[J]. *IEEE Access*, 2016, 4: 5896–5907. doi: 10.1109/ACCESS.2016.2597169.

[24] ZHANG Lianhong, ZHOU Wenqi, XIA Junjuan, *et al*. DQN-based mobile edge computing for smart Internet of vehicle[J]. *EURASIP Journal on Advances in Signal Processing*, 2022, 2022(1): 45. doi: 10.1186/s13634-022-00876-1.

[25] WANG Jin, HU Jia, MIN Geyong, *et al*. Dependent task offloading for edge computing based on deep reinforcement learning[J]. *IEEE Transactions on Computers*, 2022, 71(10): 2449–2461. doi: 10.1109/TC.2021.3131040.

[26] SUTTON R S and BARTO A G. Reinforcement Learning: An Introduction[M]. 2nd ed. Cambridge: A Bradford Book, 2018: 47–50.

[27] LIU Y C and HUANG Chiyu. DDPG-based adaptive robust tracking control for aerial manipulators with decoupling approach[J]. *IEEE Transactions on Cybernetics*, 2022, 52(8): 8258–8271. doi: 10.1109/TCYB.2021.3049555.

[28] HU Shihong and LI Guanghui. Dynamic request scheduling optimization in mobile edge computing for IoT applications[J]. *IEEE Internet of Things Journal*, 2020, 7(2): 1426–1437. doi: 10.1109/JIOT.2019.2955311.

李志华：男，教授，硕士生导师，研究方向为边缘计算、云计算与云数据中心理论、大数据挖掘、计算成像、信息安全等.

余自立：男，硕士生，研究方向为边缘计算.

责任编辑：余 蓉